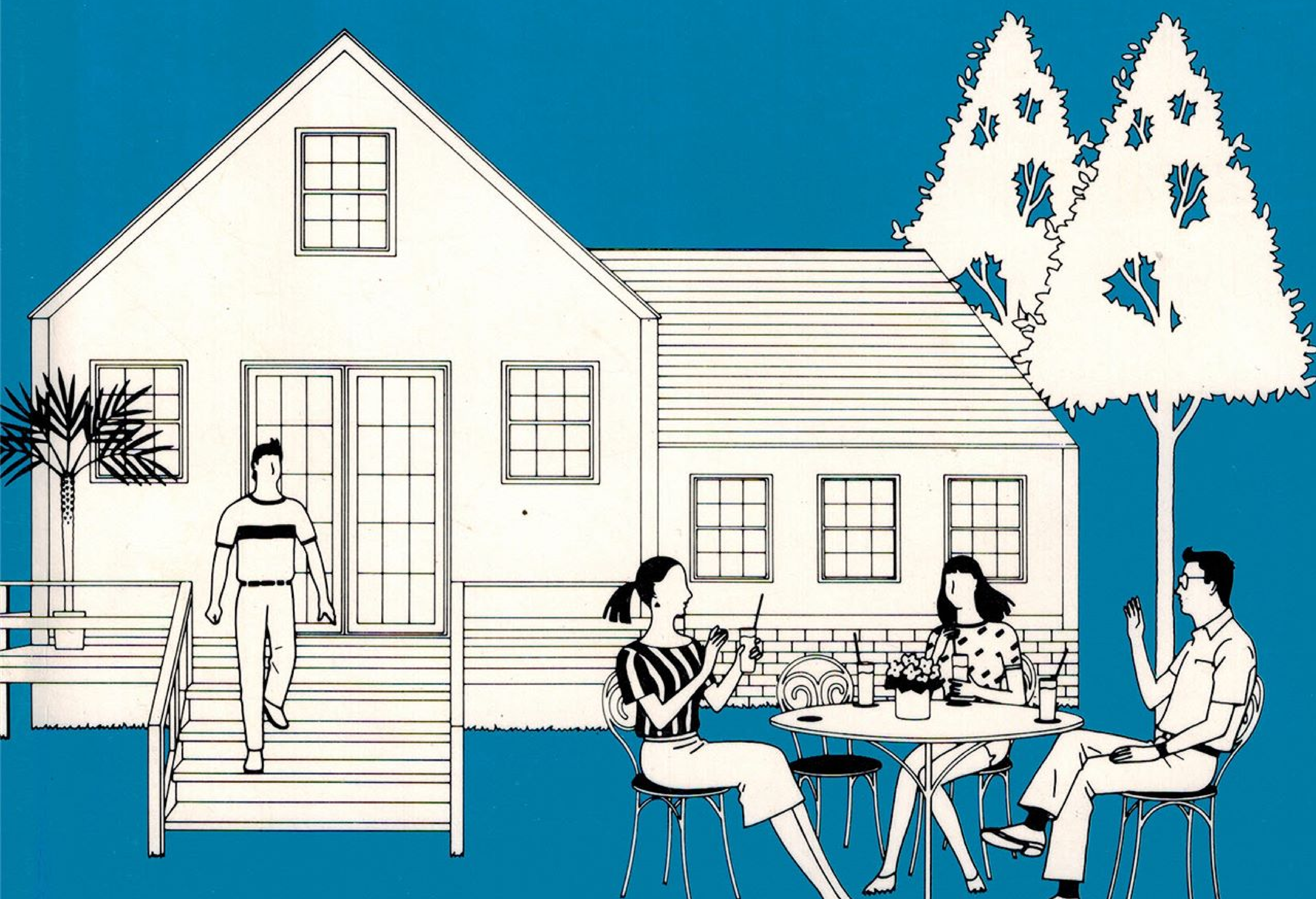
NECパーソナルコンピュータ
PC-9800シリーズ

NEC

# PC-9801UV21

## BASICリファレンスマニュアル



PC-9801UV21-RM

AA742A

# PC-9801UV

## BASICリファレンスマニュアル

# はじめに

PC-9800シリーズパーソナルコンピュータには標準プログラミング言語として$N_{88}$-BASIC(86)が用意されています.

$N_{88}$-BASIC(86)言語で記述されたプログラムは,本体ROMあるいは各種インタフェースボードに搭載されている$N_{88}$-BASIC(86)インタプリタにより解釈/実行されます.

また,$N_{88}$-日本語BASIC(86)システムディスクを使用することにより,ROMに搭載されている$N_{88}$-BASIC(86)インタプリタの持つ機能に加え,次のような機能が利用できます.

- ディスクファイルに対する入出力
- 文節変換入力あるいは表示選択入力方式による日本語入力
- 中間色表示が可能な高速グラフィック機能(4096色中の任意の8色あるいは16色が選べます)
- 倍精度数学関数
- プログラム連結機能
- モデム-NCU内蔵電話機制御機能

本書は,$N_{88}$-BASIC(86)言語が持つ機能をコマンド・ステートメント/関数単位に解説した説明書(文法書)です.

$N_{88}$-BASIC(86)あるいは$N_{88}$-日本語BASIC(86)インタプリタの操作法,動作環境あるいは$N_{88}$-BASIC(86)言語機能の応用的な説明,具体的なプログラミングの方法に関しては,本体に本書とともに添付されている「ユーザーズマニュアル」を参照してください.

また,サウンド制御命令あるいはモデム-NCU内蔵電話機制御命令に関しても「ユーザーズマニュアル」を参照してください.

なお,GP-IBインタフェース制御関連命令に関しては,GP-IBインタフェースボードに添付されているユーザーズマニュアルを参照してください.

# 目　次

## 第3章　関　　数

## APPENDIX

# 機能別索引

## コマンド



## 一般命令

## テキスト画面制御



## グラフィック画面制御

## 算術関数



## エラー制御



## 1バイト系文字操作

## 日本語文字列操作



## ファイル制御

## ライトペン制御



## I/O ポート入出力制御



## カセットテープ制御



## メモリ管理

## 機械語プログラム制御



## モード変更



## ディスク入出力

# この説明書の構成

本書は次の3章およびAPPENDIXから構成されています.

### 第1章　$N_{88}$-BASIC(86)言語の概要

この章では$N_{88}$-BASIC(86)言語の基本的な規則について解説しています.

### 第2章　コマンド・ステートメント

### 第3章　関数

第2章および第3章においては命令の機能を個々に解説しています.

### APPENDIX

エラーメッセージ一覧, キャラクタコード, 予約語等の付加情報が記載されています.

# 第1章

# $N_{88}$-BASIC(86)言語の概要

## 1.1　文

文とはプログラムの最小構成単位です．

文は１つの命令から成ります．命令は命令の名前と命令に与える情報（オペランド）から成ります．オペランドには定数，変数，関数等を指定することができます．

```
PRINT      1234
命令の名前  オペランド
      命令
       文
```

## 1.2　行

行は一般的に１つの文からなります．

行の先頭には行番号をつけます．１行には254文字の範囲で文の記述が可能です．

１行には複数の文を記述することができます．１行に複数の文を記述する場合，各文をコロン（：）で区切っていきます．これは複文（マルチステートメント）と呼ばれ，１行の範囲内で任意の数の複文が記述できます．

## 1.3　行番号

行番号は１から65529までの整数で指定します．

行番号はプログラムをメモリに格納したりディスクにセーブしたりする場合の格納順序を示すものです．

プログラムの実行は行番号の若い方から行われます．また，分岐（プログラムの実行順序を変えること)や，プログラム編集のときにも使用されます．

**備考**：行に行番号を付ける場合と付けない場合とでその動作が異なります．
行番号がなく，いきなり文が記述されている場合，リターンキーの入力によりその文は直ちに実行されます．
先頭に行番号が記述されている場合，その行はプログラムを構成する１つの行としてメモリに格納されます．

## 1.4　キャラクタセット

使用できる文字は次のとおりです．

**英大文字，英小文字，数字，カナ，特殊記号，グラフィックキャラクタ，日本語文字**

なお英大文字と英小文字は，特定の場所(たとえば文字定数の中…文字定数に関しては「1.6.2 文字定数」を参照してください）をのぞき，同じ意味に解釈されます．

## 1.5 特殊記号の使い方

演算子（+，−，＊，／）などの他にも特別な意味を持つ記号があります．ここでその意味をまとめて説明しておきます．

1. ピリオド(．)

   最近に処理の対象となった行番号の値を示す記号として使用することができます．たとえば，新しい行を挿入した，エラーが発生したなどの行です．

   例）　LIST .

2. マイナス記号(−)

   LIST，DELETE 命令など行の範囲を指定する時，何行から何行までという場合に使います．算述演算子の負号と同じ記号です．

   例）　DELETE 100−200

3. コロン(：)

   マルチステートメントの区切りとして使います．

   例）　A＝B+C : PRINT A

4. コンマ(，)

   パラメータが並ぶ場合その区切りとして使用されます．

   例）　INPUT A，B，C

   　　　COLOR 7,,,0

5. セミコロン(；)

   PRINT 文などの区切りとして使います．

   例）　PRINT "A＝" ; A

6. アポストロフィ(’)

   REM 文（注釈文）の代用として使えます．

7. クエッションマーク(？)

   PRINT 文の代用として使えます．

8. アスタリスク(＊)

ラベル名の先頭に付けます.

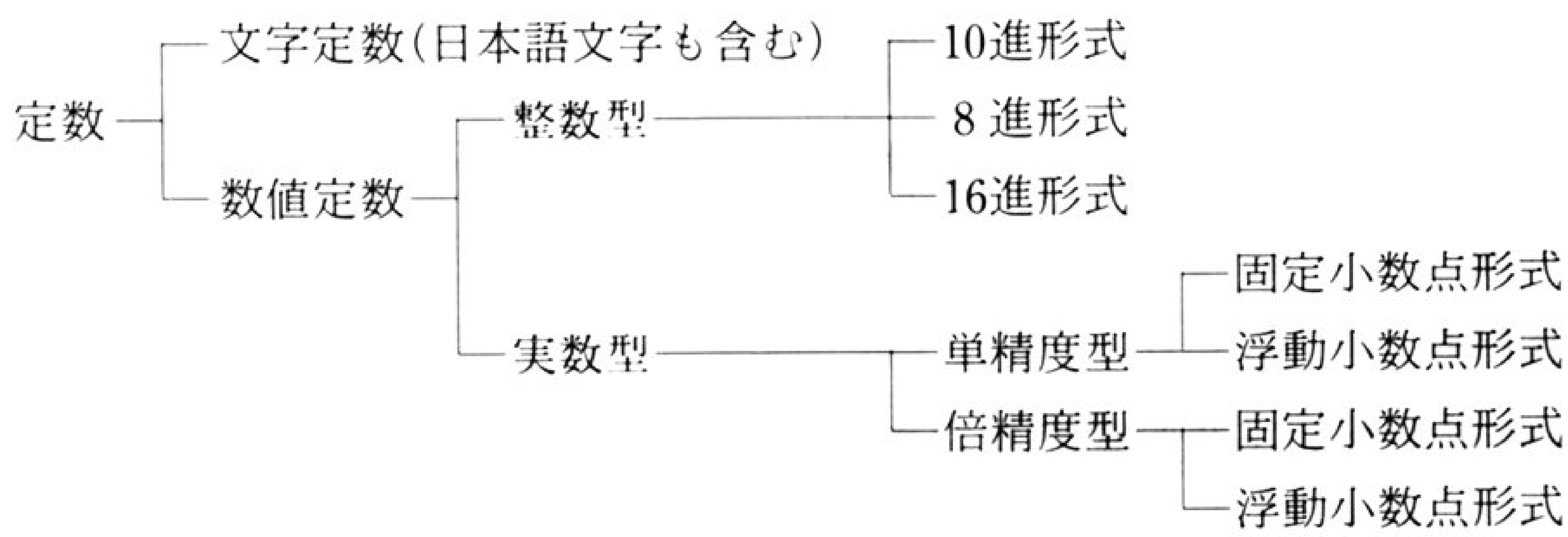
9. スペース

原則として文字定数に含まれているスペース以外のスペースは無視されます. ただし命令の名前の直後には必ずスペースを入れなければなりません.

## 1.6 定　　数

### 1.6.1 定数の種類

定数には以下のものがあります.




### 1.6.2 文字定数

文字定数とは, 255文字以下のダブルクォーテーション(″)で囲まれた英数字, カナ文字, 記号ならびに日本語文字などの列のことです. なお, (″)を文字定数中に直接記述することはできません. (″)を文字定数としたい場合, 例(5)のように特殊な方法を用います.

例)　(1) 〝Good Morning″

　　(2) 〝１２３４５６７８９″

　　(3) 〝パーソナル　コンピュータ″

　　(4) 〝日本語″

　　(5) CHR$(&H22)＋〝１２３４″＋CHR$(&H22)

### 1.6.3 数値定数

数値定数は, 整数型, 実数型に分けられ, それらは正あるいは負の数, または0です.

負の数の前に必ず符号をつけなければなりませんが, 正の数の前の符号は省略することができます.

### 1.6.4 整 数 型

(1) 10進形式

−32768から＋32767までのすべての整数，または，数の後に％をつけたもの．小数点をつけることはできません．

例)　32767

−123

32767％←──整数を表す．

(2) 8進形式

頭に&Oまたは&を伴った，0から7までの数字の並び．

&0〜&177777の範囲

例)　&12345

&O7777

(3) 16進形式

頭に&Hを伴った，0からFまでの並び．

例)　&H100

&HFFFF

**注意：** 8進形式または16進形式で入力された数値は，PRINT，LPRINTでは10進形式で出力されます．

### 1.6.5 実 数 型

実数型は，単精度型と倍精度型に分けられます．

### 1.6.6 単精度型

有効桁7桁の精度で格納されます．出力のときは7桁目が四捨五入され，6桁以下で表示されます．扱える数値は，−1.70141E＋38〜1.70141E＋38です．

(1) 7桁以下の実数

(2) Eを使った指数形式

(3) 最後に！を伴った数

例)　1.23

−7.09E−06

3525.68

3.14！

### 1.6.7 倍精度型

有効桁16桁の精度で格納され，16桁以下で表示されます．

(1) 8桁以上の数

(2) Dを使った指数形式

(3) 最後に#を伴った数

例） 1234567890

−1.09432D−06　+0.3141592653D+01

56789.0#

8657036.1543976

## 1.7 変　数

### 1.7.1 変　数

プログラム中で使われる値を格納するためのエリアに，英数字から成る名前を対応させたものが変数です．

変数の値は，プログラマによって定義され，演算，参照などに使うことができます．数値変数は，値が割り当てられるまえに参照されたら初期値として0が割り当てられ，文字変数の場合，ヌルストリング（空の文字列）が割り当てられます．

### 1.7.2 変数名と型宣言文字

変数名は最大40文字の英字で始まる英数字とピリオド（.）で表され，そのすべてを区別します．

例1） 次の2つの変数は異った変数名として解釈されます．

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890123A

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890123B

変数名は予約語（APPENDIX E「予約語」を参照してください．）であってはなりませんが，予約語を含んだものは構いません．ただし，FNで始まる変数名は許されません．また英文字において大文字，小文字の区別はありません．

同じ変数名でも型が違えば区別されます．変数の型は型宣言文字によって決まり，型宣言文字は変数名の最後につけて，その変数の型を表します．型宣言文字を省略すると，〝!〞がついているとみなされます（単精度実数型変数となる）．

型宣言文字
- %　整数型
- !　単精度実数型
- #　倍精度実数型
- $　文字型

例2）　A
　　　A#
　　　A%　これらは区別されるが，AとA！は同じ．
　　　A$

変数の型を宣言するのにもう1つ便利な方法があります．それはDEFINT, DEFSNG, DEFDBL, DEFSTRの型宣言文をプログラム中で用いる方法です．これについては2章で詳しく説明します．

### 1.7.3　配列変数

配列とは，1つの変数名でいくつかの要素を参照することのできる変数です．配列変数のそれぞれの要素は，整数または整数表記による添字によって参照されます．

配列変数の次元は1行(255文字)中に記述できる範囲内で設定することができ，添字はメモリの範囲内で許されており，これらの大きさはDIM文(2章参照)で宣言します．ただし，各添字は原則として0から始まりますから，実際の要素数は添字の数＋1となります．

例)　DIM A(10)　　　1次元配列，要素の数＝11
　　DIM TA(10, 50)　　2次元配列，要素の数11×51＝561
　　DIM TTA$(2, 5, 3)　　3次元配列，要素の数3×6×4＝72

注：各添字の数が10以下の時はDIM文による宣言を省略することができます．

### 1.7.4　予約変数

次の変数は，ユーザーによって一般の変数として使うことはできません．

TIME$　　現在の時分秒をHH：MM：SSの形でもっています．同じ形で代入することもできます．

DATE$　　現在の年月日をYY／MM／DDの形でもっています．同じ形で代入することもできます．

MID$　　文字列変数の一部に文字列を代入することができます．

ERL　　エラーの生じた時，エラーが発生した行番号をもっています．代入することはできません．

ERR　　エラーが発生した時，生じたエラーのエラーコードをもっています．代入することはできません．

## 1.8 型変換

数値データは，必要に応じてその型から他の型に変換することができます．ただし，文字型と数値型の間でこの変換を行うことはできません．

(1) ある型の数値データが，違った型の数値変数に代入された場合，数値は，その変数名によって宣言された型に変換されます．

例)
```
10 ABC%=1.234
20 PRINT ABC%
RUN
1
```

(2) 精度の違う数値間の演算の場合，精度の高い方に変換されて，演算が行われます．たとえば，10#／3の場合，10#／3#として演算が行われます．

例)
```
10 A#=10#/3
20 B#=10#/3#
30 PRINT A#, B#
RUN
  3.333333333333333     3.333333333333333
```

(3) 論理演算の場合，扱われる数値はすべて整数に変換され，結果は整数で与えられます．

例)
```
10 A=12.34
20 B=NOT A
30 PRINT B, A
RUN
-13               12.34
```

(4) 実数が整数に変換される場合は，小数点以下は四捨五入されます．この時，整数型で扱える範囲を超えた場合はエラーが起こります．

例)
```
10 A%=34.4
20 B%=34.5
30 PRINT A%, B%
RUN
  34           35
```

```
10 A#=1.234E+07
20 B%=A#
30 PRINT B%, A#
RUN
Overflow in 20
```

(5) 倍精度変数が単精度変数に代入された時は，変数の値は有効数字7桁に丸めたものとなります．単精度変数の精度は7桁であり，もとの倍精度の数値との誤差の絶対値は，5.96E－8以下となります．

例）

```
10  A#=1.23456789#
20  B!=A#
30  PRINT  A#, B!
RUN
  1.23456789        1.23457
```

倍精度変数（あるいは倍精度定数）と単精度変数（あるいは単精度定数）を混合して演算したり，単精度の値を倍精度変数に代入したりすると，単精度値に変換する際，有効桁以降の桁に変換誤差が混入しますので注意してください．

例）

① 精度の異なる数値による演算（演算結果に変換誤差が混入する）

好ましくない例

A#=1.41421356#+0.12

次のようにすればよい．

A#=1.41421356#+0.12#

② 精度の高い変数に対する精度の低い値の代入

好ましくない例

A#=3.1415

次のようにすればよい．

A#=3.1415#

## 1.9 式と演算

式とは定数や変数を演算子で結合した一般的な数式や，単に文字や数値，あるいは変数だけのものをいいます．

例）

```
"BASIC"
3.14
10+3/5
A+B/C-D
TAN(DO)
```

BASICの演算は5つに分類されます．

1. 算術演算
2. 関係演算
3. 論理演算
4. 関数
5. 文字列演算

### 1.9.1 算術演算

算術演算子には次のようなものがあります．

| 実行順序↓ | 演算子 | 演算 | 例 |
|---|---|---|---|
| | ∧ | 指数演算 | X∧Y |
| | − | 負号 | −X |
| | ＊，／ | 乗算，実数の除算 | X ＊ Y，X／Y |
| | ＋，− | 加算，減算 | X＋Y，X−Y |

演算の実行順序を変更する場合カッコを使用します．カッコの中の演算子は他の演算より先に実行されます．カッコ内においては通常の実行順序に従います．

次に実行例を示します．

| | 代数表記 | BASIC の表記 |
|---|---|---|
| 1) | $2X+Y$ | 2＊X＋Y |
| 2) | $\frac{X}{Y}+2$ | X／Y＋2 |
| 3) | $\frac{X+Y}{2}$ | (X＋Y)／2 |
| 4) | $X^2+2X+1$ | X∧2＋2＊X＋1 |
| 5) | $X^{Y^2}$ | X∧(Y∧2) |
| 6) | $(X^Y)^2$ | X∧Y∧2 |
| 7) | $Y(-X)$ | Y ＊−X |

#### (1) 整数の除算と剰余の計算

整数の除算は¥によって行われます．扱われる数値が実数の場合は演算が実行される前に小数点以下が四捨五入されます．商は小数点以下が切り捨てられた整数となります．

例)　10¥3＝3　　(10／3＝3・・・1)
　　　23.75¥5＝4　　(24／5＝4・・・4)

剰余の演算は MOD によって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます．結果は整数の割り算の余りです．

例)　13.3　MOD　4＝1　　(13／4＝3・・・1)
　　　25.68　MOD　6.99＝5　　(26／7＝3・・・5)

#### (2) 0での除算

式の実行時に 0 での除算が行われた場合は，エラーメッセージを出力しますが，結果は計算機が扱うことのできる最大の数とみなし，それを除算の結果として処理を続行します．

また，べき乗の実行時に，0に対して負のべき乗を行った場合も同様になります．

例）

```
PRINT 2／0
／0
 1.70141E+38
```

```
PRINT 0∧−1
／0
 1.70141E+38
```

(3) **桁あふれ（オーバーフロー）**

代入や演算の結果がその変数の型内で表現することのできる範囲をこえた場合，桁あふれが発生します．

桁あふれが起こった場合，〝Overflow〟エラーを出力し，最大の数を結果として与え，処理を続行します．

例）

```
PRINT 3∧300
OV
 1.70141E+38
```

## 1.9.2 関係演算

関係演算子は2つの数値を比較するときに用います．結果は，真(−1)，偽(0)で得られ，条件判定文などプログラムの流れを変えるのに用いられます（IF 文参照）．

| 関係演算子 | 内容 | 例 |
|---|---|---|
| ＝ | 等しい | X＝Y |
| <>，>< | 等しくない | X<>Y，X><Y |
| < | 小さい | X<Y |
| > | 大きい | X>Y |
| <＝，＝< | 小さいか等しい | X<＝Y，X＝<Y |
| >＝，＝> | 大きいか等しい | X>＝Y，X＝>Y |

**注意**：＝は代入文にも使うので注意すること．

IF 文の中での使い方の例を次に示します．

```
IF  X=0  THEN 1000
IF A+B< >0 THEN X=X+1 : Y=Y+1
```

## 1.9.3 論理演算

論理演算子は複数の条件を調べたり，ビット操作やブール演算を行ったりするのに用います．論理演算は，ビットごとに0または1を結果として与えます．

各論理演算の内容を次に示します.

NOT = not (否定)

| X | NOT X |
|---|---|
| 1 | 0 |
| 0 | 1 |

AND = and (論理積)

| X | Y | X AND Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

OR = inclusive or (論理和)

| X | Y | X OR Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

XOR = exclusive or (排他的論理和)

| X | Y | X XOR Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

IMP = implication (包含)

| X | Y | X IMP Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

EQV = equivalence (同値)

| X | Y | X EQV Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

論理演算子も関係演算子のように，プログラムの流れを変えるのに用いられます．この場合，論理演算子は2つ以上の関係演算子と結ぶことができます．

例) (1) IF X<0 OR 99<X THEN 1000
(2) IF 0<X AND X<100 THEN X=0
(3) IF NOT (A=0) THEN 20

(1) Xが負または，99より大きければ，行番号1000へ飛ぶ．
(2) Xが正でかつ，100より小さければ，Xに0を代入する．
(3) Aが0でなければ，行番号20へ飛ぶ．

論理演算子は演算の前に扱う数値を −32768から +32767までの2の補数表示の整数に変換します．もし，この範囲外となれば〝Overflow〟エラーとなります．もし，0(偽)と−1(真)しか与えられなかったなら，論理演算子は0と−1しか結果として与えません．

指定された論理演算では，この整数に対しビットごとに演算を行います．したがって，論理演算子はバイトデータをあるビットパターンに照らし合わせて調べることができます．

たとえば，AND 演算子は機器のI／Oポートのステータスバイトの必要なビット以外のすべてのビットをマスクするのに使えます．また OR 演算子はある2進数を作るために2つのビットパターンを混合させることができます．

以下の例は論理演算子がどのように働くかの例です．

63　AND　8=8　　$63=(111111)_2$，$8=(001000)_2$

したがって，63　AND　$8=(001000)_2=8$

−1　AND　8=8　　$-1=(1111111111111111)_2$，$8=(001000)_2$

したがって，−1　AND　8=8

12　OR　11=15　　$12=(1100)_2$，$11=(1011)_2$

したがって，12　OR　$11=(1111)_2=15$

32767　OR　−32768=−1

$32767=(0111111111111111)_2$

$-32768=-(1000000000000000)_2$

したがって，32767　OR　$-32768=(111111111111111)_2=-1$

12　XOR　11=7　　$12=(1100)_2$，$11=(1011)_2$

したがって，12　XOR　$11=(0111)_2=7$

10　XOR　10=0　　$10=(1010)_2$

したがって，10　XOR　$10=(0000)_2=0$

−X=NOT X−1　　任意の数の負数はその数の1の補数に1を加えたもの（2の補数）である．

### 1.9.4 関　　数

関数とは与えられたある引数に対して，ある決った演算を行うもので，値としては，この演算の結果を返します．

$N_{88}$-BASIC(86) には SIN（正弦），SQR（平方根）などの数値関数や CHR$，LEFT$などの文字列関数等豊富な〝組み込み関数〞が用意されています．これらの関数については3章で詳しく説明します．

また，〝ユーザー定義関数〞としてユーザーが自由に定義できる関数機能もあります．これは2章の〝DEF FN〞の項で説明します．

また，これらの初等関数（SIN 関数など）は，引数が倍精度のときは倍精度となり，引数が整数や単精度のときは単精度となります．

例)
```
A=SIN(3.14)+COS(3.14)
PRINT 2, 2*2, SQR(2)
```

## 1.10 文字列の演算

### 1.10.1 文字列の連結

文字列は演算子＋によって連結することができます．

例)
```
10 A$= "NEC" : B$= "PERSONAL" : C$= "COMPUTER"
20 D$=A$+ " "+B$+ " "+C$
30 PRINT D$
RUN
NEC PERSONAL COMPUTER
```

### 1.10.2 文字列の比較

文字も数値の比較に用いられるものと全く同じ関係演算子を用いて比較することができます．

=, <, >, <>, ><, <=, =<, >=, =>

文字列の場合それぞれの文字列の最初から1文字ずつ，文字の比較を行います．もし相互に全く同じ文字列の場合は，その2つの文字列は等しくなりますが，1箇所でも違った場合は，その文字のキャラクタコード（APPENDIX C のキャラクタコード参照）の大きい方の文字列が大きくなります．文字列の片方が短かくて比較が途中で終った場合は，短かい文字列の方が小さくなります．

文字列の比較においては空白なども意味をもちますから注意してください．

例)
```
"AA" < "AB"
"BASIC" = "BASIC"
"X&" > "X #"
"PEN " > "PEN"
"cm" > "CM"
"DESK" < "DESKS"
```

このように，文字列の比較は文字列の内容を調べたり，文字をアルファベット順に並べたり（ソート）することに使うことができます．

## 1.11 演算の優先順位

演算は次の順に処理されます．

1. カッコで囲まれたもの
2. 関数
3. 指数（べき乗）∧

4. 負号（－）
5. ＊，／
6. ¥
7，MOD
8. ＋，－
9. 関係算演子（<，>，＝など）
10. NOT
11. AND
12. OR
13. XOR
14. IMP
15. EQV

## 1.12　ラベル

プログラムの流れを変える命令として，GOTO 文，IF 文等の命令がありますが，その飛び先の指定は一般的に行番号を使用します．

次のプログラムはその一例です．

```
10  INPUT A
20  IF A<0 THEN 80
30  IF A>0 THEN 60
40  PRINT “zero”
50  GOTO 90
60  PRINT “plus”
70  GOTO 90
80  PRINT “minus”
90  END
```

これは，10行で入力された値が正か負かゼロかを調べるプログラムです．ここで処理の流れを変える命令が20，30，50，70行に使われていて，その飛び先の指定はすべて行番号になっています．しかし，この行番号はあくまで数字の並びであり，我々にはなじみにくく，プログラムの作成，デバッグ時において混乱を招くことがあります．

ラベルを使用することにより，このような問題が解決されます．ラベルとはどのようなものかを説明する前に，まずそのラベル名を使って先程のプログラムを書き換えてみましょう．

```
10  INPUT A
20  IF A<0 THEN *MINUS
30  IF A>0 THEN *PLUS
40  PRINT "zero"
50  GOTO *EXIT
60  *PLUS : PRINT "plus"
70  GOTO *EXIT
80  *MINUS : PRINT "minus"
90  *EXIT : END
```

このようにラベル名とは，分岐先の目印として独自に付けることのできる名前なのです．

このラベル名の使用においては，次の注意を守らなければなりません．

1. ラベル名の頭には必ずアスタリスク(*)を付けなければなりません．
2. 先頭のアスタリスクを除いて，ラベル名は必ず英文字で始まらなければなりません．
3. ラベル名に使用できる文字は，先頭のアスタリスクを除いて英文字と数字とピリオド(.)であり，大文字と小文字の区別はありません．
4. 予約語(PRINT など)をラベルとして使用することはできません．ただし，中に含む場合は構いません．
5. ラベル名の長さは，プログラムの1行の許可範囲（1行255文字）によってのみ制限を受けます．
6. 参照されるラベル名(呼ばれる方のラベル名)は，必ず行の最初になければなりません．
7. 1行中でラベル名の後に続けて命令を記述し，マルチステートメントとする時は，コロン(：)またはスペースによって区切ります．

以上のような制限を無視すると "Syntax error" となります．

その他参照されるラベル名に同じものがあった場合，2重定義されているという意味の "Duplicate label" エラーが生じます．これらのエラーの検出は，インタプリタの場合他のエラーメッセージの場合と異なり，"RUN" した時，プログラムの実行に先だって行われますから，ラベル名のエラーがあるとプログラムを1ステップも実行することができません．

また，この場合のエラーメッセージはエラー箇所を示す〈行番号〉は伴いません．

ラベル名は，プログラム中で分岐の目印として使用する他，"LIST"，"DELETE" など〈行番号〉を対象とする文のパラメータにはすべて使用することが可能です．

例)
```
LIST *START-*LAST
DELETE *LOOP1-*EXIT1
```

# 第2章・第3章の見方

2章・3章では個々の命令について書式，機能等を説明しています．

各命令の説明は次の構成で行われています．

**命令の名前**　(DISKモード)

| 機　能 |
|---|

| 書　式 |
|---|

| 文　例 |
|---|

| 解　説 |
|---|

| サンプルプログラム |
|---|

(DISKモード) $N_{88}$-日本語BASIC(86) システムディスクからシステムを立ち上げた場合(このような環境をDISKモードと呼びます)のみ有効な機能であることを意味します．この機能は，ROMに搭載されている $N_{88}$-BASIC(86)を使用している場合には使うことができません．

機　能　命令の機能を簡単に示します．

書　式　命令の記述の仕方を示します．実際の入力時には次のような決まりに従ってください．

1. アルファベットの大文字で示された項目は，そのまま入力します．入力する場合は，小文字でも大文字でもかまいません．ただし，ダブルクォーテーション (") で囲まれた文字列 (ファイル名以外) は，大文字と小文字を区別してください．
2. カギカッコ〝<〞,〝>〞で囲まれた項目は，ユーザーが指定します．
3. 角カッコ〝〔〞,〝〕〞で囲まれた項目は，オプションであり省略することができます．省略した場合，デフォルト値(BASICであらかじめ設定されている値) または以前に指定した値が適用されます．

   コンマ (,) で区切られる複数のパラメータがあり省略可能な場合，次の例のような書式で示しています．

   **CONSOLE 〔<スクロール開始行>〕〔,<スクロール行数>〕〔,<ファンクションキー表示スイッチ>〕〔,<カラー／白黒スイッチ>〕**

   この場合，角カッコで囲まれた内容の意味は，パラメータを省略することが可能であることを示しています．そこで，以後のパラメータがすべて省略された場合は，コンマも含めて省略しますが，後に続くパラメータを指定する場合は，それ以前のコンマはすべて指定する必要があります．

たとえば，上記の例で〈カラー／白黒スイッチ〉のみを指定する場合，次のように指定します．

CONSOLE　,,, 1

4. 上記のカギカッコ，角カッコ以外の記号でカッコ（　），コンマ（,），セミコロン（；），マイナス記号（－），等号（＝）などの記号は示された位置に正しく入力します．
5. 省略記号〝・・・〞の続く項目は，1行の許す長さ（255文字）の内で任意の回数繰り返すことができます．

   **例：　〈定数〉〔,〈定数〉・・・〕**の場合　0，10，15
6. 座標指定の所で（Wx，Wy）と記されているのはワールド座標を，（Sx，Sy）はスクリーン座標を表しています．その他単に（X，Y）と記されているのはキャラクタ座標を表しています．またSTEP（x，y）という記法は相対座標による指定ができることを意味しています．
7. 書式の中で〈行番号〉またはそれに類して行番号の指定をさしているものは，行番号そのものの他にラベル名を含んでいると解釈してください．

**文　例**　実際の入力の仕方の見本として簡単な例を示します．

**解　説**　命令の使用方法や詳しい機能とそれに関して注意しなければならない点などを説明します．

**サンプルプログラム**　いくつかの文を交えたプログラム例とその実行結果を示します．

3章〝関数〞の算術関数の中で，初等関数（sin，cos，tanなど）は引数の値として倍精度型の数値を与えると倍精度型で，その他の数値を与えると単精度型でその値を返します．ただし，倍精度関数機能はDISKモードでのみ使うことができます．

# 第2章

## コマンド・ステートメント

# AUTO



機 能　行番号を自動的に発生します.

書 式　AUTO〔〈行番号〉〕〔,〈増分〉〕

文 例　AUTO 100, 5

解 説　AUTOコマンドは, 指定された〈行番号〉を発生し, 行の入力待ちになります. 以後, リターンキーの入力ごとに, 〈増分〉で指定された増分ずつ増加した行番号を自動的に発生します. 〈行番号〉, 〈増分〉ともに省略された場合には, 10が省略値として選択されますが, コンマがあって〈増分〉が省略された場合には, 直前に実行したAUTOコマンドの増分が用いられます.

CTRL-Cまたは, ストップキーを押すと, AUTOコマンドはその動作を中止し, BASICのコマンドレベルに戻ります. この時, 最後に発生された行番号の行は格納されません.

プログラム中に既に存在する行と同じ行番号が発生された場合には, 行番号の直後にアスタリスク(*)が表示され, 注意を促します. この時, 文字を入力してリターンキーを押すと, その行は新しい内容に入れ代わります. また, 何も入力しないでリターンキーを押した場合には, その行は削除されます.

AUTOのモード中においても, カーソルエディッティング機能を利用することができます. カーソルを移動して既に入力した行, あるいは表示されていた行のところで, リターンキーを入力した場合には, その行の行番号に増分を足した行番号が次に発生される行番号となります.

# BEEP

| | |
|---|---|
| 機　能 | **内蔵スピーカを鳴らします.** |
| 書　式 | BEEP〔<スイッチ>〕 |
| 文　例 | BEEP |

解　説

内蔵スピーカを鳴らしたり，止めたりします．<スイッチ>の値が1の場合はON（鳴りっぱなし），0でOFFとなります．

また，<スイッチ>を省略した場合には，PRINT CHR$(7)を実行したのと同様に，一定時間スピーカを鳴らします．

サンプルプログラム

```
100 '--bell--
110 FOR I=1 TO 20
120  BEEP 1 : FOR II=1 TO 100 :NEXT
130  BEEP 0 : FOR II=1 TO 100 :NEXT
140 NEXT I
150 BEEP
```

# BLOAD　　(DISK モード)



**機　能**　機械語プログラムをメモリ上にロードします.

**書　式**　BLOAD 〈ファイルディスクリプタ〉〔, 〈ロードアドレス〉〕〔, R〕

**文　例**　BLOAD "SUB1", R

**解　説**　〈ファイルディスクリプタ〉によって指定された, 機械語プログラムファイルをメモリ上にロードします.

〈ロードアドレス〉が省略された場合, BLOAD によるロードは直前に実行された DEF SEG 文で指定されたセグメントベースに, BSAVE により機械語プログラムファイルをセーブする際に指定した相対アドレスを加えた番地から行われますが, 〈ロードアドレス〉が指定された場合には, 〈ロードアドレス〉で与えられた番地からロードし始めます. したがって, 〈ロードアドレス〉を付けてロードするプログラムは, セーブされた際と異なる番地にロードされても実行可能な性質(リロケータブル)を持っていなければなりません.

Rオプションを指定すると, プログラムをロード後, 機械語プログラムの先頭からプログラムの実行を直ちに開始します. この時, 既に開かれているファイルはその状態を保持します. 〈ロードアドレス〉が指定されている場合は, 実行開始番地も DEF SEG 文で指定されたセグメントベースに 〈ロードアドレス〉 を加えた番地になります.

**参照**：BSAVE, CLEAR, DEF SEG, LOAD

**対象ファイル**：ディスク, RS-232C コミュニケーションファイル

**サンプルプログラム**

```
        .
        .
        .
100 CLEAR ,&H3E00
110 DEF SEG=&H3E00
120 BLOAD "SUB1",R
        .
        .
        .
```

# BSAVE　(DISK モード)

機　能　機械語プログラムをディスクファイルあるいはRS-232Cコミュニケーションファイルにセーブします.

書　式　BSAVE 〈ファイルディスクリプタ〉, 〈開始番地〉, 〈長さ〉

文　例　BSAVE "SUB1", &H100, &H2FF

解　説　メモリ上に置かれている機械語プログラムを，〈ファイルディスクリプタ〉で指定されたファイルにセーブします．〈開始番地〉は，相対アドレスで指定し，直前に実行されたDEF SEG文で指定された機械語プログラムセグメントのベースアドレスにこの値を加えた番地以後の連続する〈長さ〉バイトの内容が機械語プログラムとしてセーブされます.

参照：BLOAD, CLEAR, DEF SEG, SAVE

対象ファイル：ディスク，RS-232Cコミュニケーションファイル

サンプルプログラム

```
        .
        .
        .
100 CLEAR ,&H3E00
110 DEF SEG=&H3E00
120 BSAVE "SUB1",0,&H100
        .
        .
        .
```

# CALL



機能 機械語サブルーチンを呼び出します.

書式 CALL 〈変数名〉〔(〈引数〉〔, 〈引数〉…〕)〕

文例 CALL MUSBR(ARG1, ARG2)

解説 メモリ中に用意された機械語サブルーチンに制御を移します.

〈変数名〉は, 機械語サブルーチンの実行開始番地を相対アドレスで指定し, 直前に実行されたDEF SEG文で指定されたセグメントベースに〈変数名〉で指定された変数に代入されている値が加えられ, 実行開始番地となります. ここで用いる変数には配列変数を用いることはできません.

〈引数〉は, 機械語サブルーチンに渡す変数を指定します. 引数としてはすべての型の変数を指定することができますが, 定数や式を渡すことはできません.

CALLによって呼び出されるサブルーチンは, 機械語のIRET命令によりBASICに制御を戻すことができます.

参照：BLOAD, BSAVE, CLEAR, DEF SEG

サンプルプログラム

```
100 CLEAR ,&H3E00
110 DEF SEG=&H3E00
120 BLOAD "SAMPLE"
130 CULC=&H0
140 WORK =&H100
150 ARG1%=10
160 ARG2%=23
170 CALL CULC(ARG1%,ARG2%)
        .
        .
        .
```

# CHAIN　(DISK モード)

| | |
|---|---|
| 機　能 | **メモリ上のプログラムとディスク上のプログラムを連結し実行します.** |
| 書　式 | CHAIN〔MERGE〕<ファイルディスクリプタ>〔, <行番号>〕〔, ALL〕〔, DELETE <範囲>〕 |
| 文　例 | CHAIN MERGE "TEST.BAS", 500, ALL, DELETE 500-600 |

解　説

<ファイルディスクリプタ>で指定されたディスクファイル内のプログラムをロードし，実行します.

メモリ上にあるプログラム中で使用中のファイルはそのままの状態で，ロードされるプログラムに引き継がれます. <行番号>は呼び出されたプログラムの実行開始行を指定するもので，省略した場合はプログラムの最初から実行します. ここで指定する行番号は，RENUM を実行しても書き換えられません. またラベル名も使うことはできません.

ALL オプションは呼び出されたプログラムに変数,配列を引き渡すのに使います. これを指定した場合すべての変数，配列が引き渡され，省略した場合いっさい引き渡されません. 必要な変数，配列だけを引き渡したいという場合は COMMON 文を使います (COMMON 参照).

MERGE オプションは，現在メモリ上にあるプログラムと指定されたプログラムを混合(マージ)し，結果のプログラムを指定した<行番号>から実行します. <行番号>が省略されている場合は，混合したプログラムの最初から実行します. ただし，MERGE オプションを指定した場合，呼び出されるプログラムは必ずアスキーセーブ(SAVE 参照)されたものでなければなりません.

DELETE オプションは MERGE オプションを指定する時のみ意味をもつもので，プログラムを混合する前にあらかじめ，メモリ上のプログラムの指定した範囲を削除します. ここで指定する行番号の範囲は，RENUM を実行することによって書き換えられます. またこの行番号の代わりにラベル名を使うこともできます.

MERGE オプションを指定した場合，DEFINT, DEFSNG, DEFDBL, DEF STR 等の宣言文は，連結されるプログラムに対しても有効となりますが，MERGE オプションの指定がない場合，これらの宣言文は連結されるプログラムに何の効果も与えません.

また，プログラムを連結すると，DEF FN あるいは ON GOSUB, ON GOTO 等の割り込み制御情報は無条件に無効となります.

なお，OPTION BASE は無条件に連結されるプログラムに引き継がれます．

**参照**：COMMON

**対象ファイル**：ディスクのみ

サンプルプログラム

**呼ぶ側（メモリ上）のプログラム**

```
100 DIM IDENT(200)
110 COMMON IDENT(),COUNT
120 OPEN "DATA" AS #1
130 FIELD #1,4 AS I$,20 AS N$,50 AS A$
140 INPUT "会員番号";I
150 IF I=0 THEN GOTO *PRINTOUT
160 IF I=>1000 THEN STOP
170 INPUT "氏名";NAME1$
180 INPUT "住所";ADR$
190 LSET I$=MKS$(I)
200 LSET N$=NAME1$
210 LSET A$=ADR$
220 PUT #1,I
230 COUNT=COUNT+1
240 IDENT(COUNT)=I
250 GOTO 140
260 *PRINTOUT
270 CHAIN "CHAIN2"
280 END
```

**呼ばれる側（ディスク内）のプログラム**

```
100 COMMON IDENT(),COUNT
110 FIELD #1,4 AS I$,20 AS N$,50 AS A$
120 IF COUNT=0 THEN STOP
130 GET #1,IDENT(COUNT)
140 PRINT "会員番号：";CVS(I$)
150 PRINT "氏名：";N$
160 PRINT "住所：";A$
170 COUNT=COUNT-1
180 GOTO 120
```

# CIRCLE

**機　能**　楕円(円)を描きます.

**書　式**　CIRCLE ｜(Wx, Wy)／STEP (x, y)｜, 〈半径〉〔, 〈パレット番号 1〉〕〔, 〈開始角度〉〕〔, 〈終了角度〉〕〔, 〈比率〉〕〔, F〔, ｜〈パレット番号 2〉／〈タイルストリング〉｜〕〕

**文　例**　CIRCLE (80, 80), 40, 4, 0, 6.28

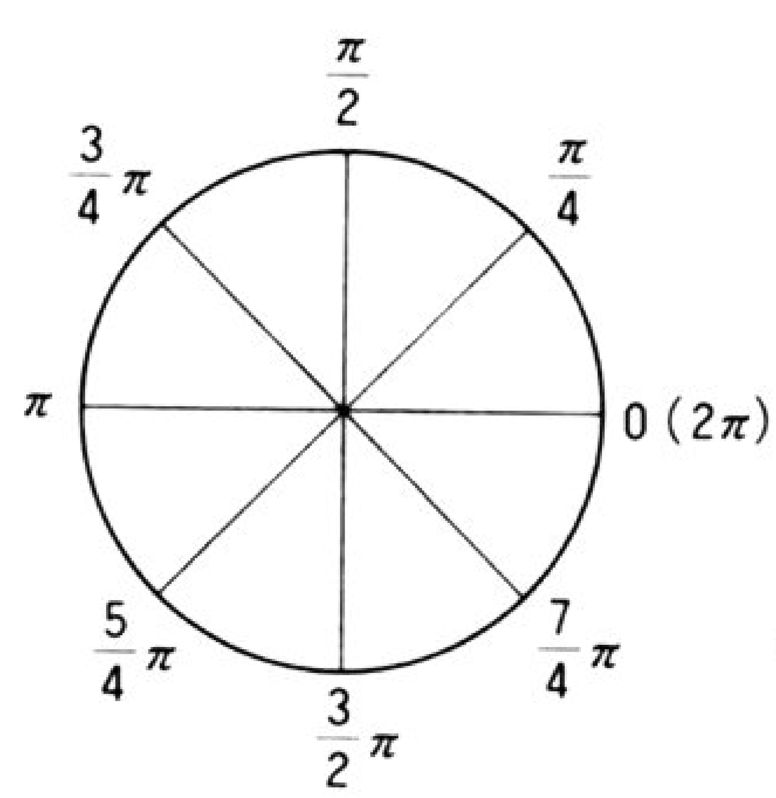
**解　説**

ワールド座標の (Wx, Wy)を中心とし, 〈半径〉で指定される大きさの円を描きます.

〈パレット番号 1〉が指定されると, 指定されたパレットの色で円を描きます. 省略された場合には, COLOR 文で指定されているフォアグラウンドカラーが用いられます.

CIRCLE は通常円を描きますが, 〈開始角度〉, 〈終了角度〉を指定すると, 指定された角度の範囲内のみに円弧を描きます. 角度の指定の単位はラジアンで, 省略値はそれぞれ, 0 と $2\pi$ です.

〈開始角度〉, 〈終了角度〉が負であった場合には, その絶対値をとり, 正にした角度が用いられますが, その時中心から半径が描かれますので扇形を描くことができます. 〈開始角度〉, 〈終了角度〉は, $-2\pi$ から $+2\pi$ の範囲内にないと "Illegal function call" のエラーとなります.




$\pi=3.14159$

〈比率〉は, (垂直方向の半径)／(水平方向の半径)で指定します. ただし, 640×400ドットのモードでは実際に表示される画面上の長さの比率で示し, 640×200ドットのモードではその1／2の値を指定します. 省略された場合には, 640×200ドットのモードでは0.5, 640×400ドットのモードでは1.0が用いられます.

〈比率〉が１以下の場合には，〈半径〉は水平方向の半径を意味します．また１以上の場合には，〈半径〉は垂直方向の半径を意味します．つまり，640×400ドットのモードの場合，〈半径〉が同じでも〈比率〉が１以下ならば平べったい楕円となり，１以上ならば長細い楕円となります．

Ｆが指定された場合，円を描くと同時にその内部を，〈パレット番号２〉で指定されたパレットの色または〈タイルストリング〉により指定された模様(タイル)でぬりつぶします．

〈パレット番号２〉および〈タイルストリング〉とも省略された場合には，円を描いたパレットの色でぬりつぶします．

開始，終了角度の指定がある場合には，扇形の内部をぬりつぶします(Ｆが指定された場合は，開始，終了角度は負であるとみなされます)．

中心座標は，STEPを付けてLP（Last referenced point）からの相対座標により指定することもできます．

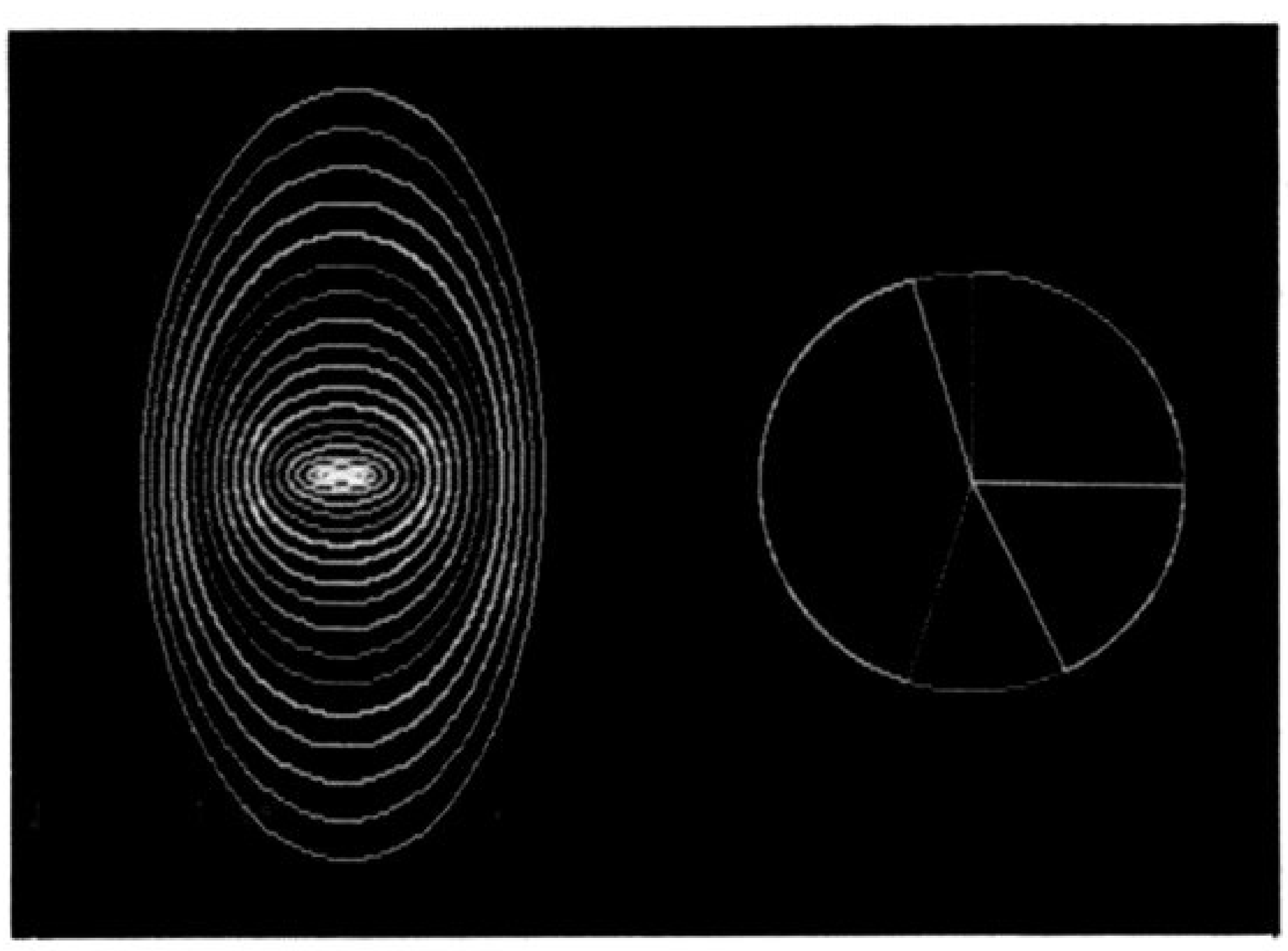
CIRCLEを実行すると，LPは円の中心座標（Wx，Wy）に設定されます．

〈タイルストリング〉については，〔２〕PAINTを参照してください．

**参照**：COLOR，〔２〕PAINT

サンプルプログラム

```
100 '-- circle display --
110 SCREEN 0,0:CLS 3
120 FOR R=1 TO 100 STEP 5
130  CIRCLE(150,100),R,(R MOD 7)+1,,,R/100
140 NEXT R
150 '
160 ST=.00001
170 DATA 25,5,40,13,17
180 FOR I=1 TO 5
190 READ DAT:EN=ST+DAT/100*3.14*2
200  CIRCLE(450,100),100,I,-ST,-EN
210  ST=EN
220 NEXT I
```



# CLEAR

**機　能**　変数の初期化およびメモリレイアウトを決定します．

**書　式**　CLEAR 〔〈ダミーパラメータ〉〕〔，〈メモリの上限〉〕〔，〈スタックの大きさ〉〕〔，〈配列データ領域の大きさ〉〕

**文　例**　CLEAR ，&H3E00

**解　説**

すべての数値変数を 0 に，また文字変数を "" (ヌルストリング) に初期化します．

〈メモリの上限〉は，16の倍数のセグメントアドレス(1)で指定し，指定された場合には，指定された番地の直前までを BASIC の使用するメモリの上限値としますので，その番地以降に置かれたデータや，機械語プログラムは，BASIC によってクリアあるいは破壊されることはありません．

注(1)：セグメントアドレスは，実際のアドレスを16で割った値．

〈スタックの大きさ〉は，BASIC が FOR，GOSUB，PAINT 等に使用するスタック領域の大きさをバイト数で指定します．リセット時の初期化された状態では512バイトに設定されています．

CLEAR は，DEF 文 (DEF FN，DEF SEG，DEF USR，DEFINT 等) によって定義あるいは指定された情報もすべて無効にします．

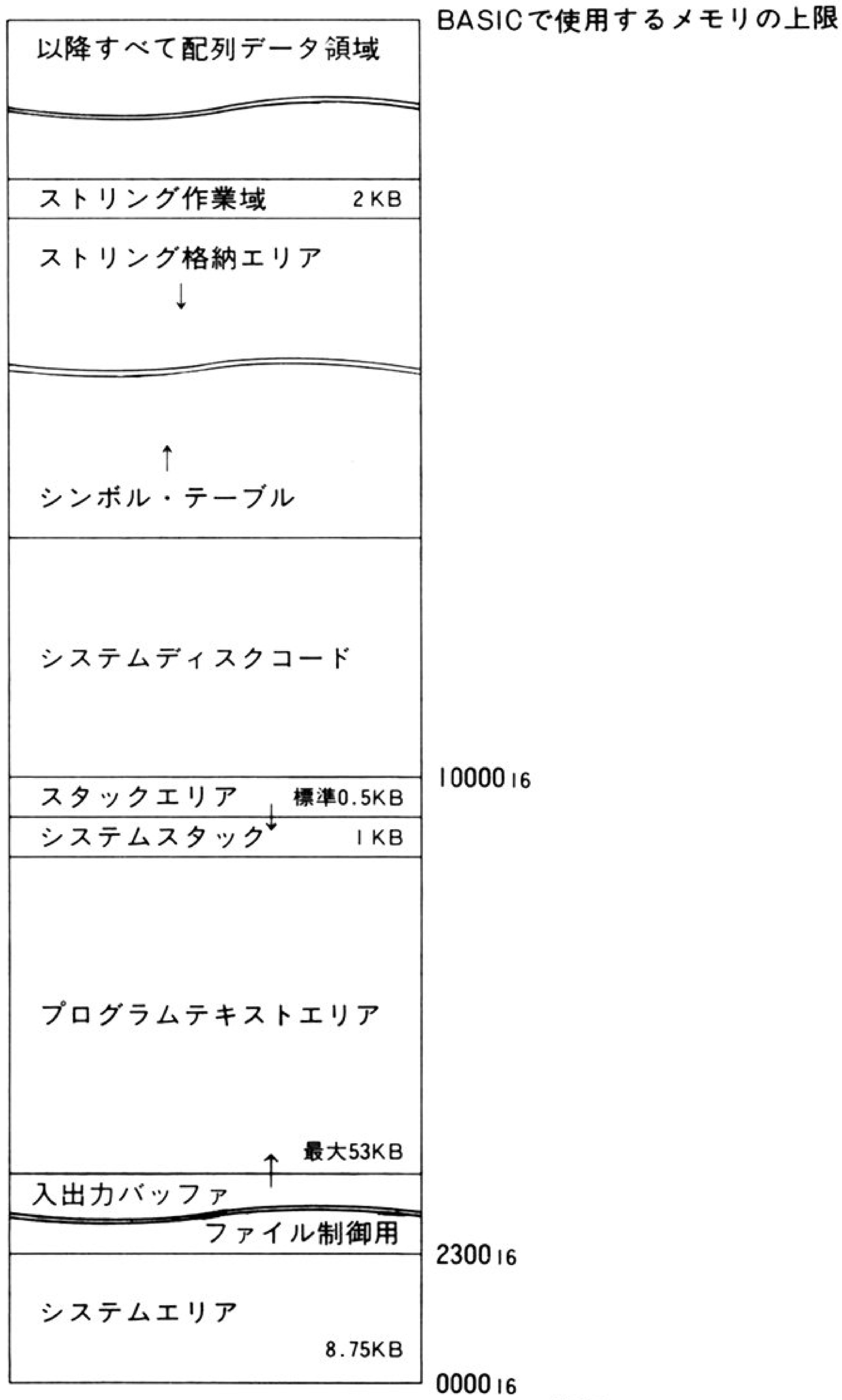
〈配列データ領域の大きさ〉は，数値配列の値が格納される配列データ領域のサイズを指定します．実際に確保される配列データ領域のサイズは指定された値を16倍したサイズです．一般にこのパラメータは，配列データ領域を縮少し，その分シンボルテーブル／ストリング格納エリアのサイズ (最大 62K バイト) を拡大するために使用します．

この値がメモリの上限より大きかったり，実際に取れないような値の場合は，"Illegal function call" エラーになります．

第 1 番目のパラメータはダミーパラメータです．値を指定しても何の効果もありませんので省略してください．

**参照**：BLOAD，BSAVE，FRE

参考：$N_{88}$-日本語 BASIC(86)のメモリレイアウト




$N_{88}$-日本語BASIC(86)のメモリレイアウト概要

メモリレイアウトの詳細については，お手持ちのユーザーズマニュアルを参照してください．

# CLOSE

**機　能**　ファイルを閉じます．

**書　式**　CLOSE〔〔#〕<ファイル番号>〔，〔#〕<ファイル番号>〕…〕

**文　例**　CLOSE

CLOSE #1，#3

**解　説**

<ファイル番号> に対応するファイルを閉じます．以後，指定された<ファイル番号>は，異なるファイルを開くために利用できるようになります．また閉じられたファイルは，同じあるいは異なったファイル番号によって再び開くことができます．

<ファイル番号>を複数指定することにより，一度に複数のファイルを閉じることができます．また<ファイル番号>が省略された場合には，その時，開いているファイルすべてを閉じます．

ファイルに対しては，再び開かれるまで入出力を行うことはできません．

ファイルが出力用に開かれていた場合には，バッファに残っていたデータの掃き出しを行いますので，ファイルの出力処理を正しく終了するためには，CLOSE 文の実行が必要です．

RUN あるいは LOAD のR指定なしはプログラムの実行を開始する前に開かれているすべてのファイルを自動的に閉じます．また，END あるいは NEW の実行によってもファイルは自動的に閉じます．STOP 文はファイルを閉じる作業は行いませんので注意してください．また，END 文が実行されずプログラムが終了する場合，すなわちプログラムの終端に END 文が存在せず，実行制御がプログラム終端に至った場合もファイルは閉じられませんので注意してください．

参照：OPEN，END，STOP，NEW，LOAD，CHAIN

# CLS

**機能** 現在アクティブな画面をクリアします．

**書式** CLS〔<機能>〕

**文例** CLS 2

**解説** <機能>は，1，2，3の値をとり，それぞれ次のように働きます．省略された場合には1が選択されます．

1：テキスト画面のみをクリアします．テキスト表示モードがCONSOLE文，COLOR文により，白黒，リバースモードになっている場合には画面は白くなります．

2：グラフィック画面のビューポート内のみをクリアします．グラフィック表示がSCREEN文によりカラーモードに設定されている場合には，COLOR文により指定されたバックグラウンドカラーで，ビューポート内をクリアします．

3：テキスト画面，グラフィック画面の両方をクリアします．

テキスト画面がクリアされる時は，CONSOLE文によって指定されたスクロールウィンドウ内のみをクリアします．またグラフィック画面をクリアした場合には，LP（Last referenced point）はビューポートの左上の頂点に移動します．

参照：SCREEN，COLOR，CONSOLE，VIEW

**サンプルプログラム**

```
100 SCREEN 0,0
110 GOSUB *DISPLAY
120 BEEP:CLS 1
130 GOSUB *DISPLAY
140 BEEP:CLS 2
150 GOSUB *DISPLAY
160 BEEP:CLS 3
170 GOSUB *DISPLAY:VIEW(200,50)-(400,150),,7
180 BEEP:CLS 3
190 SCREEN 0:END
200 '-- display routine --
210 *DISPLAY:FOR I=1 TO 20
220  LOCATE RND*78,RND*23
230  PRINT "*"
240  LINE(RND*639,RND*199)-STEP(40,20),INT(RND*7+1),B
250 NEXT I:RETURN
```

# 〔1〕 COLOR

**機能**　**ディスプレイ画面の各部の色およびグラフィック画面のパレットモードを指定します．**

**書式**　COLOR〔<ファンクションコード>〕〔,<バックグラウンドカラー>〕〔,<ボーダーカラー>〕〔,<フォアグラウンドカラー>〕〔,<パレットモード>〕

**文例**　COLOR 7，0，0，7

**解説**　テキスト画面の文字のカラーを変えたり，グラフィック画面のフォアグラウンド，バックグラウンド，ボーダーのカラーを設定したり，グラフィック画面のパレットモードを指定する機能があります．

〈ファンクションコード〉は，テキスト画面の文字にいろいろな機能を与えます．このファンクションコードは，現在テキスト画面がカラーモードになっているか，白黒モードになっているかによって働きが異なります．

**白黒モードの場合（CONSOLE ,,,0）**

0――ノーマル（通常の表示）
1――シークレット（文字は表示されない）
2――ブリンク（点滅する）
3――シークレット（1と同じ）
4――リバース（反転する）
5――リバースシークレット（反転して文字は表示されない）
6――リバースブリンク（反転して点滅する）
7――リバースシークレット（5と同じ）

**カラーモードの場合（CONSOLE,,,1）**

| | | | |
|---|---|---|---|
| 0――黒 | 1――青 | 2――赤 | 3――紫 |
| 4――緑 | 5――水色 | 6――黄色 | 7――白 |

ただし，テキスト画面の文字の色は，あくまでこのカラーコードによって表されます．〔2〕COLORで説明される〝パレット番号〞ではありません．したがってパレットを変化させても文字色が変化することはありません．

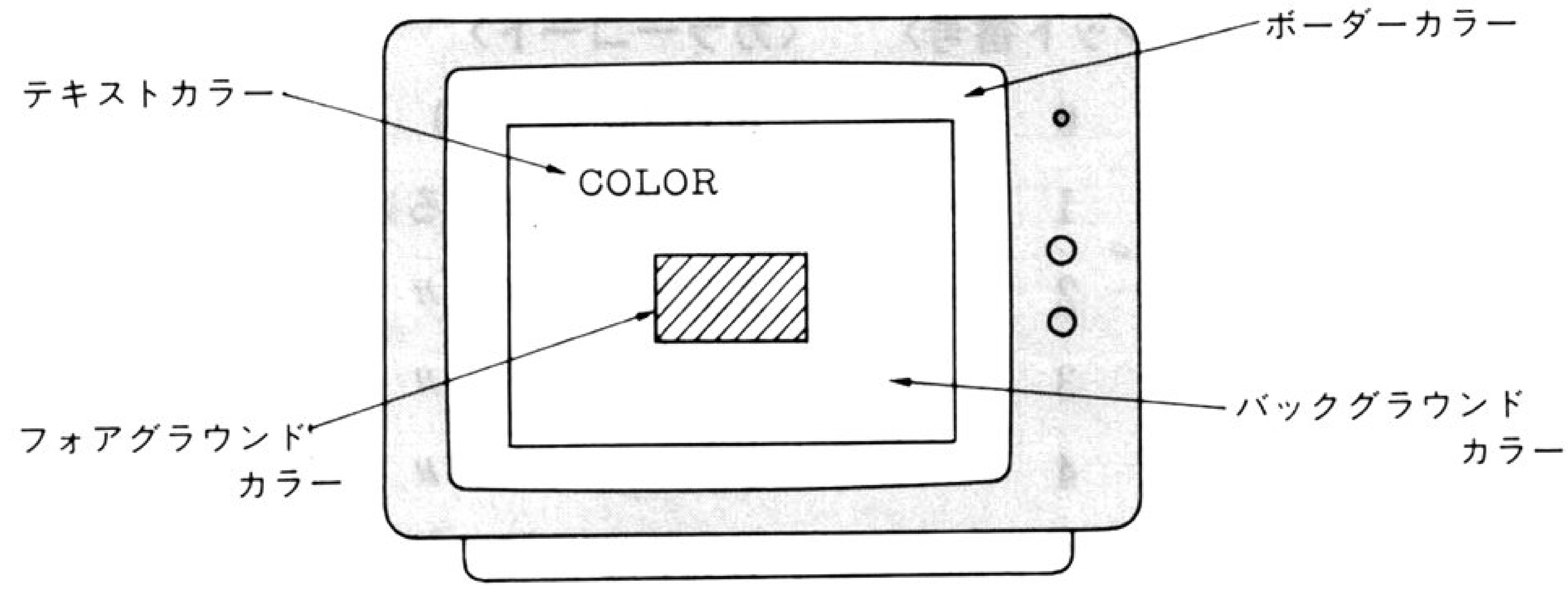
〈バックグラウンドカラー〉と〈フォアグラウンドカラー〉はグラフィック画面のカラーを設定するもので，〝パレット番号〞によって指定します．

〈フォアグラウンドカラー〉とは，グラフィック画面に点や線を表示したりする時に使われる色のことです．種々のグラフィック命令（PSET，LINE，CIRCLE

など）で特別に色指定をしなかった場合，このフォアグラウンドカラーが採用されます．

〈バックグラウンドカラー〉とは，グラフィック画面の地の色のことで，この命令実行後CLS命令によって画面をクリアすると，この色によって画面が塗り変えられます．また以後PRESET命令を色指定なしで実行すると，この色が採用されることになります．

〈ボーダーカラー〉とは，ディスプレイのBASICによって使うことのできる領域外の色のことです．ボーダーカラーは，パレット番号ではなくカラーコードで指定します．したがってテキスト画面のカラーと同じく，パレットを変化させても色が変化することはありません．ただし，専用高解像度ディスプレイ使用時は，このパラメータを指定して色を付けることはできません．




〈パレットモード〉はグラフィック画面に対する色指定のモードを指定します．

〈パレットモード〉に指定できる値とその意味は次の通りです．

0：8個のパレットにシステムが決めた8色を対応づけるモードです（このモードを8色中・8色モードと呼びます）．

1：8個のパレットに4096色中の任意の8色を対応づけるモードです（このモードを4096色中・8色モードと呼びます）．

2：16個のパレットに4096色中の任意の16色を対応づけるモードです（このモードを4096色中・16色モードと呼びます）．

$N_{88}$-BASIC (86)が立上がった直後のパレットモードは8色中・8色モードです．以降，〈パレットモード〉が指定された場合に限りパレットモードが切り変わります．

各モードにおけるパレットとカラーコードの対応づけの方法に関しては〔2〕COLORを参照してください．

〈パレットモード〉がある場合，パレットモードが切り変わり，パレットとカラーコードの関係は次のように初期化されます．

8色中・8色モード

| 〈パレット番号〉 | | 〈カラーコード〉 | |
|---|---|---|---|
| 0 | ←―― | 0 | (黒) |
| 1 | ←―― | 1 | (明るい青) |
| 2 | ←―― | 2 | ( 〃 赤) |
| 3 | ←―― | 3 | ( 〃 紫) |
| 4 | ←―― | 4 | ( 〃 緑) |
| 5 | ←―― | 5 | ( 〃 水色) |
| 6 | ←―― | 6 | ( 〃 黄色) |
| 7 | ←―― | 7 | (白) |

4096色中・8色モード

| 〈パレット番号〉 | | 〈カラーコード〉 | |
|---|---|---|---|
| 0 | ←―― | &H000 | (黒) |
| 1 | ←―― | &H00F | (明るい青) |
| 2 | ←―― | &H0F0 | ( 〃 赤) |
| 3 | ←―― | &H0FF | ( 〃 紫) |
| 4 | ←―― | &HF00 | ( 〃 緑) |
| 5 | ←―― | &HF0F | ( 〃 水色) |
| 6 | ←―― | &HFF0 | ( 〃 黄色) |
| 7 | ←―― | &HFFF | (白) |

4096色中・16色モード

| 〈パレット番号〉 | | 〈カラーコード〉 | |
|---|---|---|---|
| 0 | ←―― | &H000 | (黒) |
| 1 | ←―― | &H00F | (明るい青) |
| 2 | ←―― | &H0F0 | ( 〃 赤) |
| 3 | ←―― | &H0FF | ( 〃 紫) |
| 4 | ←―― | &HF00 | ( 〃 緑) |
| 5 | ←―― | &HF0F | ( 〃 水色) |
| 6 | ←―― | &HFF0 | ( 〃 黄色) |
| 7 | ←―― | &HFFF | (白) |
| 8 | ←―― | &H777 | (灰色) |
| 9 | ←―― | &H00A | (少し暗い青) |
| 10 | ←―― | &H0A0 | ( 〃 赤) |
| 11 | ←―― | &H0AA | ( 〃 紫) |

| | | | | |
|---|---|---|---|---|
| 12 | ←—— | &HA00 | （ 〃 | 緑） |
| 13 | ←—— | &HA0A | （ 〃 | 水色） |
| 14 | ←—— | &HAA0 | （ 〃 | 黄色） |
| 15 | ←—— | &HAAA | （ 〃 | 白） |

**注意**：〈パレットモード〉はDISKモードでのみ有効なパラメータです（ROMモードで〈パレットモード〉の指定を行うとエラーになります）.

また，4096色中・8色モードおよび4096色中・16色モードはアナログRGB対応ディスプレイが接続されている場合のみ有効です（エラーにはなりませんがアナログRGB対応ディスプレイが接続されていない場合，指定通りの色は出ません）.

**参照**：〔2〕COLOR，COLOR@，CONSOLE

サンプルプログラム

```
100 CONSOLE ,,,1
110 SCREEN 0,0
120 COLOR 7,4,,7:CLS 3
130 GOSUB *LINEB
140 FOR I=1 TO 7
150  COLOR I,,,I:GOSUB *LINEB
160 NEXT I:END
170 *LINEB:PRINT "COLOR"
180 LINE(100,50)-STEP(200,100),,BF
190 RETURN
```

## 〔2〕 COLOR

**機能**　カラーパレットを変更します．

**書式**　COLOR〔＝(〈パレット番号〉，〈カラーコード〉)〕

**文例**　COLOR＝(2，4)

**解説**　グラフィック画面への色の指定はすべてカラーパレットによって行います．この命令は，どのパレットにどの色を対応させるかを決めるものです．

〈パレット番号〉とは0から7(あるいは0～15)の8個(あるいは16個)のカラーパレットにつけられた固有の番号で，それぞれのパレットにどの色を対応づけるかを〈カラーコード〉によって指定します．

パレットの個数およびカラーコードの値はパレットモードによって異なります(パレットモードに関しては〔1〕COLORを参照してください)．

各パレットモードにおける〈パレット番号〉と〈カラーコード〉の関係は次の通りです．

**8色中・8色モード**

| 〈パレット番号〉 | | 〈カラーコード〉 |
|---|---|---|
| 0 | | 0 (黒) |
| 1 | | 1 (青) |
| 2 | | 2 (赤) |
| 3 | | 3 (紫) |
| 4 | | 4 (緑) |
| 5 | | 5 (水色) |
| 6 | | 6 (黄色) |
| 7 | | 7 (白) |

（どのパレットにどのカラーコードを対応させるか任意に決めることができます．）

〈カラーコード〉：の8種類中の任意の8個（同じカラーコードを複数のパレットに対応づけてもかまいません．）

**4096色中・8色モード**

| 〈パレット番号〉 | | 〈カラーコード〉 |
|---|---|---|
| 0 | | &H000 |
| 1 | | ～ |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | &HFFF |

（どのパレットにどのカラーコードを対応させるか任意に決めることができます．）

〈カラーコード〉：の4096色中の任意の8個（同じカラーコードを複数のパレットに対応づけてもかまいません．）

**4096色中・16色モード**

| 〈パレット番号〉 | 〈カラーコード〉 |
| --- | --- |
| 0 | &H000 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | &HFFF |

（どのパレットにどのカラーコードを対応させるかを任意に決めることができます.）

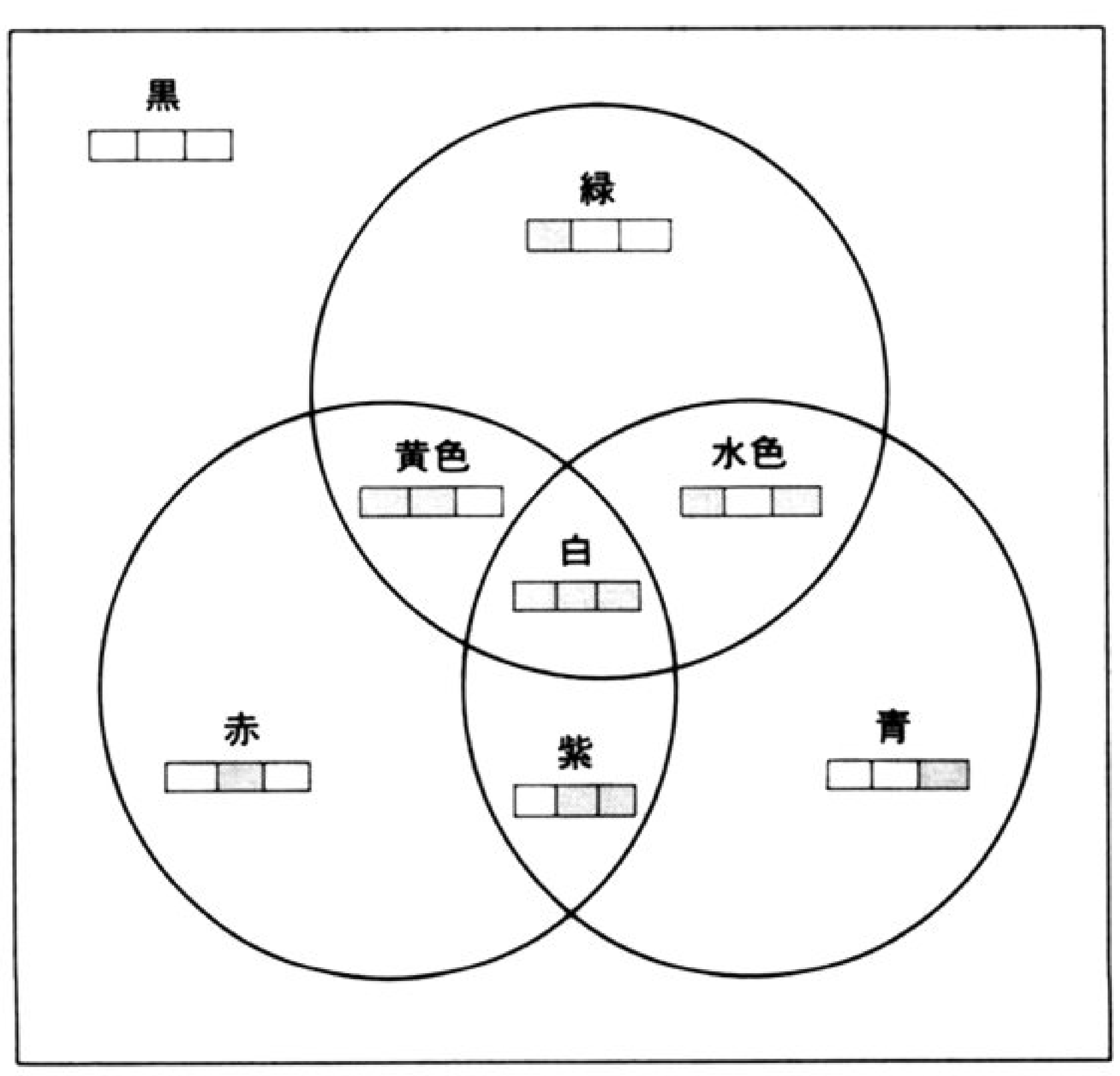
&H000～&HFFFの4096色中の任意の16個

（同じカラーコードを複数のパレットに対応づけてもかまいません.）

ここで，4096色中・8色モードおよび4096色中・16色モードにおけるカラーコードの値と色の関係について説明します.

(1) 色のしくみ

グラフィック画面に表示できる色はすべて基本色である緑(G)，赤(R)，青(B)の組み合わせで表現されます.



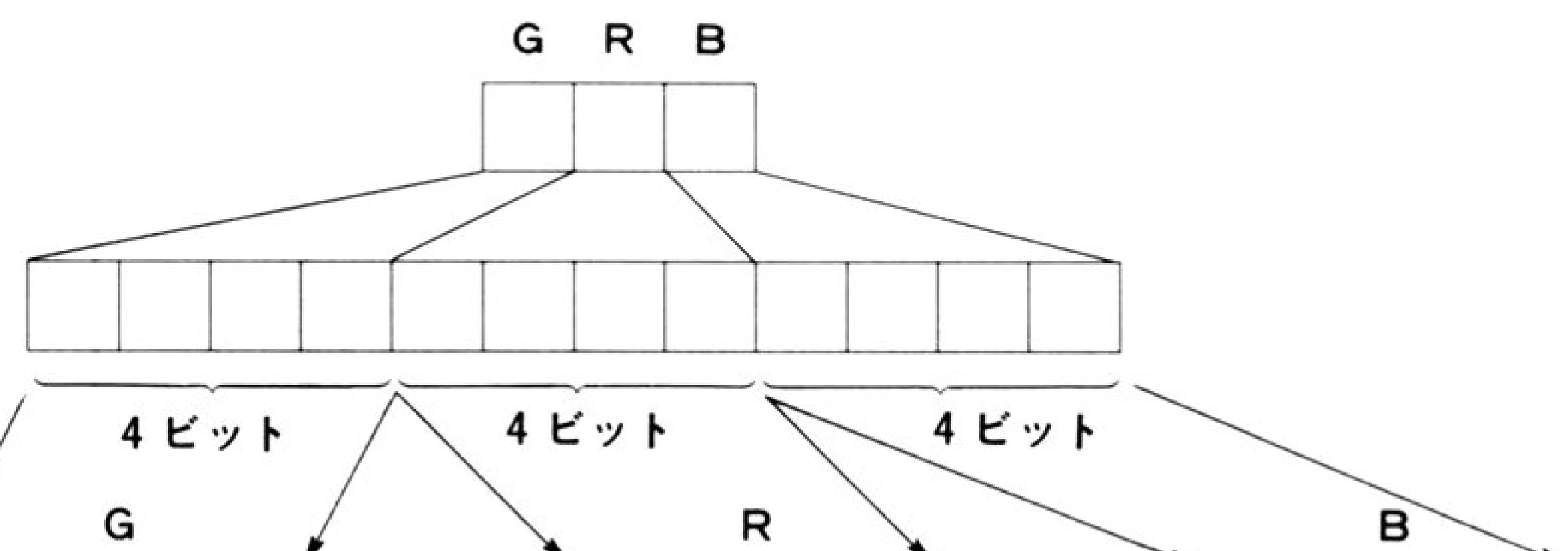

G R B □□□：左から緑(G)，赤(R)，青(B)に対応しています. ■はONの状態を，□はOFFの状態を示しています.

たとえば，水色の場合 G R B ■□■ は，BとGがONの状態です．すなわち，水色は青と緑を混ぜて表現される色であることを示しています.

(2) 4096種類の色

黒，青，赤，紫，緑，黄色，水色，白の色のしくみは(1)で示した通りですが，それではオレンジ色や黄緑などの中間色はどのように表現するのかを説明します．たとえば，オレンジ色というのは，黄色に赤を混ぜた色です．黄色は赤と緑を混ぜた色ですが，赤と緑を混ぜあわせる際に緑よりも多く赤を混ぜ合わせるとオレンジ色になります（赤と緑を同じ程度に混ぜると黄色になります）．

色を混ぜ合わせる割り合いは，「輝度」によって調整することができます．「輝度」はその名の通り明るさの度合いを示したものです．G，R，Bそれぞれ16段階の輝度を表現することができます．G，R，Bはそれぞれ4ビットの情報を持ち，このビットの状態で輝度を表わしています（16段階の輝度を表わすのには16進表現が便利です）．




暗い ↓ 明るい

G

| 輝度 | ビットの状態 |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

R

| 輝度 | ビットの状態 |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

B

| 輝度 | ビットの状態 |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

暗い ↓ 明るい

輝度が0というのは，その色が表示されないことを意味します．オレンジ色を表現する場合は，輝度が&HFの赤（最も明るい赤）と輝度が&H9の緑（暗い緑）を混ぜ合わせることによって表現します（カラーコードには&H9F0と指定します）．このように輝度を変えることで，様々な色合いが表現できます．また，基本の7色についてもその輝度を変えれば，暗い赤や，暗い黄色などの表現ができます．

例）　明るい黄色──→輝度が &HFの赤と &HFの緑の混ぜ合わせ
　　　　　　　　　　（カラーコードは &HFF0となります）
　　　暗い黄色　──→輝度が &H7の赤と &H7の緑の混ぜ合わせ
　　　　　　　　　　（カラーコード は &H770となります）

緑(G)と赤(R)と青(B)はそれぞれ16段階の輝度が表現できます．そして，これらの色の混ぜ合わせを自由に行うと16×16×16＝4096種類の色が表現できます．

パレットとカラーコードの対応を各パレットモードにおける初期状態にリセットすることができます．

次に示すように，〝＝〟（等号）も含めて〈パレット番号〉および〈カラーコード〉を省略します．

COLOR

(DISK モードでのみ有効な機能です.）

各パレットモードにおけるパレットの初期状態に関しては，〔１〕COLOR のパレットモードの説明を参照してください．

参照：〔１〕COLOR

サンプルプログラム

8 色中・8 色モードの例

```
100 '-- palette alternation --
110 COLOR ,,,,0
120 FOR I=1 TO 7
130  COLOR=(I,7)
140 NEXT I
150 '
160 FOR I=1 TO 7
170  LINE(0,I*20)-STEP(639,10),I,BF
180 NEXT I
190 '
200 FOR I=1 TO 7
210  FOR J=1 TO 7
220   COLOR=(J,I)
230    GOSUB *WAITSUB
240  NEXT J
250 NEXT I
260 '
270 '-- palette initialize --
280 COLOR
290 END
300 '-- wait routine --
310 *WAITSUB
320  FOR K=0 TO 100
330  NEXT K
340 RETURN
```

4096色中・16色モードの例

```
100 '-- palette alternation --
110 SCREEN 3,0:COLOR ,,,,2
120 FOR I=1 TO 15
130  COLOR=(I,&HFFF)
140 NEXT I
150 '
160 FOR I=1 TO 15
170  LINE(0,I*20)-STEP(639,10),I,BF
180 NEXT I
190 '
200 FOR I=1 TO 15
210  FOR J=1 TO 15
220   COLOR=(J,16*I^2)
230    GOSUB *WAITSUB
240  NEXT J
250 NEXT I
260 '
270 '-- palette initialize --
280 COLOR
290 END
300 '-- wait routine --
310 *WAITSUB
320  FOR K=0 TO 100
330  NEXT K
340 RETURN
```

# COLOR@

| | |
|---|---|
| 機　能 | **テキスト画面に書かれた文字などに色や機能を設定します.** |
| 書　式 | COLOR@($X_1$, $Y_1$)－($X_2$, $Y_2$)〔, 〈ファンクションコード〉〕 |
| 文　例 | COROR@(0, 0)－(79, 4), 2 |

解　説

テキスト画面のキャラクタ座標の2点($X_1$, $Y_1$), ($X_2$, $Y_2$)を対角とする四角形の領域に書かれている, 文字やグラフィックキャラクタに色を付けたり, ブリンクなどの機能を設定したりします.

指定する座標($X_1$, $Y_1$), ($X_2$, $Y_2$)は必ずキャラクタ座標でなくてはなりません. また〈ファンクションコード〉は, CONSOLE文によって設定されているモードによって持つ意味が異なります. この機能及び指定の仕方は〔1〕COLOR文の〈ファンクションコード〉の場合と全く同様ですから, そちらを参照してください.

もし〈ファンクションコード〉が省略された場合は, 7を値として採用します.

**注意**：この機能はあくまでテキスト画面に書かれている文字などに対して有効です. したがって何も書かれていない場合は, 何の作用もありません. またこの命令によって設定された領域の上に新しく文字などを書いた場合, 書かれた文字はこの命令の影響を受けません. なお, 白黒モードで指定された領域に表示されているグラフィック画面の表示がある場合, 指定された色に変わります.

**参照**：CONSOLE, 〔1〕COLOR

サンプルプログラム

```
100 CONSOLE 0,25,0,1
110 WIDTH 80,25:CLS
120 FOR I=0 TO 31
130  PRINT "COMPUTER  ";
140 NEXT I
150 FOR I=0 TO 7
160  COLOR@(I*10,0)-(I*10+9,4),I
170 NEXT I
```

# COMMON　(DISK モード)

機　能　CHAIN 文が実行された時，連結されるプログラムに変数を引き渡します．

書　式　COMMON <変数名>〔, <変数名> …〕

文　例　COMMON A, B, XY( ), NA$

解　説　CHAIN 文が実行され，メモリ上のプログラムとディスクからロードされたプログラムが連結された時，メモリ上のプログラムの変数を引き渡します．したがって COMMON 文は必ず CHAIN 文と合わせて使われます．

COMMON 文は CHAIN 文の前にある必要があります．

<変数名>は 1 行の許す範囲で何個でも並べることができますが，1 つのプログラム中の COMMON 文に同じ変数名があってはいけません．また配列変数名は〝( )″を付け加えて表現します．

すべての変数を引き渡したいのであれば，CHAIN 文の ALL オプションを使う方が便利です．

参照：CHAIN

サンプルプログラム　呼ぶ側（メモリ上）のプログラム

```
100 DIM IDENT(200)
110 COMMON IDENT(),COUNT
120 OPEN "DATA" AS #1
130 FIELD #1,4 AS I$,20 AS N$,50 AS A$
140 INPUT "会員番号";I
150 IF I=0 THEN GOTO *PRINTOUT
160 IF I=>1000 THEN STOP
170 INPUT "氏名";NAME1$
180 INPUT "住所";ADR$
190 LSET I$=MKS$(I)
200 LSET N$=NAME1$
210 LSET A$=ADR$
220 PUT #1,I
230 COUNT=COUNT+1
240 IDENT(COUNT)=I
250 GOTO 140
260 *PRINTOUT
270 CHAIN "CHAIN2"
280 END
```

呼ばれる側（ディスク内）のプログラム

```
100 COMMON IDENT(),COUNT
110 FIELD #1,4 AS I$,20 AS N$,50 AS A$
120 IF COUNT=0 THEN STOP
130 GET #1,IDENT(COUNT)
140 PRINT "会員番号：";CVS(I$)
150 PRINT "氏名：";N$
160 PRINT "住所：";A$
170 COUNT=COUNT-1
180 GOTO 120
```

# COM ON／OFF／STOP

| | |
|---|---|
| 機　能 | **通信回線からの割り込みの許可，禁止，停止を制御します.** |
| 書　式 | 1) COM〔(<回線番号>)〕ON<br>2) COM〔(<回線番号>)〕OFF<br>3) COM〔(<回線番号>)〕STOP |
| 文　例 | COM ON<br>COM(2) ON |

解　説

　コミュニケーションポート（RS-232C 回線）に外部からの通信が入ったことによる割り込みを許可（ON）するか，禁止（OFF）するか，停止（STOP）するかを宣言します.

　〈回線番号〉に指定できる値と意味は次のとおりです.

1 ：RS-232C 第 1 回線
2 ：RS-232C 第 2 回線 ｝専用の拡張インターフェースボードが必要です.
3 ：RS-232C 第 3 回線 ｝

　〈回線番号〉は省略することができます. 省略された場合，第 1 回線が指定されたものとみなします.

**書式 1）** 命令実行直後に割り込みを許可します. この命令実行後は，〈回線番号〉で指定したコミュニケーションポートに受信が入るごとに割り込みを発生し，ON COM GOSUB 文で定義されている処理ルーチンに分岐します.

**書式 2）** 割り込みを禁止します. この命令実行後は，通信があっても処理ルーチンへの分岐は起こりません.

**書式 3）** 割り込みを停止します. この命令実行後は，通信があってもそのことを覚えているだけで処理ルーチンへの分岐は起こりません. しかし，その後 COM ON によって割り込みが許可されると，先程の通信によって処理ルーチンに分岐します.

**注意：** プログラム終了時には COM OFF の状態にしておいてください.
COM ON 文は，RS-232C コミュニケーションポートを OPEN した後実行されねば有効となりません.

なお，割り込み処理ルーチンに制御が移ると自動的に割り込み停止状態となりますので，割り込み処理ルーチンの先頭ではCOM OFFあるいはCOM STOP命令を実行しないでください．

**参照**：ON COM GOSUB

# CONSOLE

**機能**　テキスト画面モードの設定を行います．

**書式**　CONSOLE〔〈スクロール開始行〉〕〔,〈スクロール行数〉〕〔,〈ファンクションキー表示スイッチ〉〕〔,〈カラー／白黒スイッチ〉〕

**文例**　CONSOLE 0, 25, 0, 1

**解説**　〈スクロール開始行〉と〈スクロール行数〉を指定することによって，画面上でスクロールする領域(スクロールウィンドウ)を指定します．

画面のクリア (CLS あるいは PRINT CHR$(12)) はこのスクロールウィンドウに対して実行されます．

〈ファンクションキー表示スイッチ〉に1を指定すると，画面最下行にプログラマブルファンクションキーに登録されている文字列を表示します．0を指定した場合，表示しません．起動直後は表示されるモードになっています．

〈カラー／白黒スイッチ〉を1に指定すると，テキスト画面をカラーモードにし，0を指定した場合には白黒モードにします．起動直後は白黒モードになっています．

# CONT

| | |
|---|---|
| 機　能 | **ストップキー入力，またはSTOP文によって停止したプログラムの実行を再開します．** |
| 書　式 | CONT |
| 文　例 | CONT |

解　説

CONTは通常デバッグのためにSTOP文と共に用いられます．STOP文やストップキー(CTRL-C)の入力によりプログラムの実行を停止し，ダイレクトモードで変数の値等を調べたり変更した後に，CONT文によって実行を再開します．

ただし，実行停止中にプログラム内容の変更を行った場合には，CONT文による継続実行はできません．

また，実行停止のタイミングによってはプログラムの実行継続ができない場合があります．

# COPY

| | |
|---|---|
| 機　能 | **画面情報のハードコピーを行います．** |
| 書　式 | COPY〔〈機能〉〕 |
| 文　例 | COPY 3 |

解　説

2種類の画面ハードコピー機能が用意されています．

(1) ROM BASICモードの場合あるいはDISK BASICモードでメモリスイッチSW6・$2^4$ ビットがOFFの場合．

〈機能〉に指定できる値は1～5で，それぞれ次のように働きます．

1：テキスト画面のみを出力します．

2：グラフィック画面のみを出力します．

3：テキスト画面とグラフィック画面を重ね合わせて出力します．この時，テキスト画面の情報はプリンタの印字体で出力されます．〈機能〉が省略された場合，3を値として採用します．

4：グラフィック画面のみを出力します．640×200モードの場合に有効なコピーです．

5：テキスト画面とグラフィック画面を重ね合わせ，さらに縦方向に縮小して出力します．

**注意**：PC-PR201系プリンタを使用する場合，メモリスイッチSW5・$2^0$ ビットをONにしてください．

(2) DISK BASICモードで，メモリスイッチSW6・$2^4$ ビットがONの場合．

〈機能〉に指定できる値は1～5で，それぞれ次のように働きます．

1：テキスト画面のみを出力します．

2：グラフィック画面のみを出力します（カラー出力の場合，白黒を反転出力します）．

3：テキスト画面とグラフィック画面を合成して出力します（カラー出力の場合，白黒を反転出力します）．〈機能〉が省略された場合，3を値として採用します．

以降の機能はカラー出力の場合のみ有効な機能です．

4：グラフィック画面のみを出力します（白黒を反転しないで出力します）．

5：テキスト画面とグラフィック画面を合成して出力します（白黒を反転

しないで出力します).

**注意**：PC-PR201系プリンタを使用する場合，メモリスイッチ SW5・$2^0$ ビットをON にしてください.

またカラー出力を行う場合，次の準備が必要です.

・ PC-PR201CL カラープリンタの接続

・メモリスイッチ SW6・$2^3$ ビットを ON にする

詳しくはお手持ちのユーザーズマニュアルを参照してください.

# DATA

**機　能**　READ 文で読み込まれる数値，文字定数を定義します．

**書　式**　DATA 〈定数〉〔, 〈定数〉 …〕

**文　例**　DATA 1, CBA, 1465

**解　説**

DATA 文は非実行文で，プログラム中のどこに置いてもかまいません．1つの DATA 文には，1行(255文字)に入るだけのデータをセットすることができます．また，1つのプログラム内には任意の数のデータ文を置くことができます．

READ 文は行番号の小さい方から順番に，DATA 文中のデータを読み込んでいきます．

〈定数〉は，任意の型の定数，即ち，整数，単精度実数，倍精度実数，文字定数のいずれでもかまいません．但し，定数式(2＊3など)は許されません．また READ 文で読み込む場合，READ 文で指定されている変数の型は対応する DATA 文の定数の型と一致しなければなりません．

DATA 文中のデータはコンマ(,)によって区切られますが，文字定数で，その文字列の先頭，または最後がコンマやピリオド(.)，意味のある空白である場合ならびに日本語文字を含む文字列はその文字定数の前後をダブルクォーテーション(″)でくくる必要があります．

参照：READ，RESTORE

**サンプルプログラム**

```
100 DATA 10,10,    NEC COMPUTER,"    日本電気"
110 READ A
120 READ B$:B=VAL("&H"+B$)
130 READ C$:READ D$
140 PRINT A:PRINT B
150 PRINT C$:PRINT D$


run
 10
 16
NEC COMPUTER
    日本電気
Ok
```

# DEF FN

| | |
|---|---|
| 機 能 | **ユーザー定義関数を定義します.** |
| 書 式 | DEF FN <**名前**> 〔(<**パラメータリスト**>)〕=<**関数の定義式**> |
| 文 例 | DEF FNA(X, Y)=X * 2+Y * 3 |

解 説

関数の名前は“FN”とこれに続く<名前>を含めたものであり,<名前>は変数名として正しい形をしたものでなければなりません.

<パラメータリスト>はその関数が呼ばれた時に,<関数の定義式>中の同じ名前の変数に対応し,この変数名は<関数の定義式>を評価する際にのみ有効となり,プログラム中に同一名の変数があっても影響はありません.また,<関数の定義式>中で使われている変数が,<パラメータリスト>中にない場合は,その変数がその時点で持っている値が使われます.

<関数の定義式>は,その関数の演算内容を記述する式で,1行の範囲に限られます.このユーザー定義関数は,数値関数,文字関数,その混在のいずれでもかまいませんが,<パラメータリスト>内の変数と,それに渡される変数とが同一の型でなければなりません.

DEF FN文は非実行文ですが,これによって定義される関数がプログラム中で呼ばれる前に実行されていなければなりません.

サンプルプログラム

```
100 DEF FNFUKURI(G,R,N)=G*(1+R)^N
110 DEF FNTANRI(G,R,N)=G*(1+R*N)
120 INPUT "元金   : ",GANKIN
130 INPUT "年利   : ",NENRI:INPUT "年数   : ",NEN
140 FGOKEI=FNFUKURI(GANKIN,NENRI,NEN)
150 TGOKEI=FNTANRI(GANKIN,NENRI,NEN)
160 F$="¥¥##########,._-"
170 PRINT "複利では";USING F$;FGOKEI
180 PRINT "単利では";USING F$;TGOKEI

run
元金   : 100000
年利   : .07
年数   : 10
複利では      ¥196,715.-
単利では      ¥170,000.-
Ok
```

D

# DEFINT／SNG／DBL／STR

機 能　変数の型宣言を行います.

書 式　DEF | INT | 〈文字の範囲〉〔, 〈文字の範囲〉…〕
　　　　　| SNG |
　　　　　| DBL |
　　　　　| STR |

文 例　DEFINT A, I-K

解 説　〈文字の範囲〉で指定された文字で始まる変数名の変数を, DEFINT では整数型に, DEFSNG では単精度実数型に, DEFDBL では倍精度実数型に, DEFSTR では文字型にそれぞれ定義します.

〈文字の範囲〉で指定される文字は英字 1 文字で, 複数個指定する場合はマイナス記号でつないでその範囲を示します.

ただし, これらによって行われた型宣言より, 型宣言文字による指定 (%, !, #, $)の方が優先されます. また, 型宣言が行われていない文字で始まる変数は, すべて単精度変数とみなされます.

サンプルプログラム

```
100 DEFINT I-M
110 DEFSNG A,B
120 DEFDBL D
130 I1=1.23:J=65.643
140 ABC=1.23:BBB=65.643
150 D=3.141592654000003#
160 D%=3.141592654000003#
170 PRINT "I1=";I1,,"J=";J
180 PRINT "ABC=";ABC,,"BBB=";BBB
190 PRINT "D=";D,"D%=";D%
200 END

run
I1= 1                            J= 66
ABC= 1.23                        BBB= 65.643
D= 3.141592654000003             D%= 3
Ok
```

# DEF SEG

| | |
|---|---|
| 機　能 | **セグメントベースアドレスを宣言します.** |
| 書　式 | DEF SEG= 〈セグメントアドレス〉 |
| 文　例 | DEF SEG=&H3E00 |

解　説

PEEK関数でメモリの一部を読み出したり，POKE命令でメモリの一部にデータを書き出したりする場合，そのメモリ空間を含むセグメントのベースアドレスをあらかじめ宣言しておかなければなりません.

同様に，機械語プログラムをメモリにロードする場合もあらかじめ機械語プログラムセグメントのベースアドレスを宣言しておかなければなりません.

このような場合，DEF SEG文を使用します.

PEEK，POKE等で指定するアドレスとは，DEF SEG文で指定したセグメントベースアドレスからの相対アドレスです.

DEF SEG文に指定するアドレスは実際のアドレスを16で割った値です.

なお，各変数や各配列のセグメントベースは，VARPTR関数により得ることができます．また配列については配列ごとに別セグメントとなっていますので，複数の配列の要素あるいは配列要素と変数にアクセスする場合には，そのつどセグメントベースを切り換えてアクセスすることが必要となります.

**注意**：DEF SEG文は，DEFとSEGの間に必ずスペースを入れなければなりません．スペースを省略してDEFSEGとすると変数として扱われます.

**参照**：BLOAD，BSAVE，CLEAR，DEF USR，POKE，PEEK，VARPTR

# DEF USR

**機　能**　**機械語で作られたユーザー関数の実行開始番地を定義します.**

**書　式**　DEF USR〔〈番号〉〕＝〈開始番地〉

**文　例**　DEF USR3＝&HF000

**解　説**　USR関数(ユーザー関数)が呼び出す機械語ルーチンの実行開始番地の設定を行います．〈番号〉は，0～9までの値で，複数のユーザー関数を用いる場合の識別を行います．したがって最大10個までのユーザー関数を準備，利用することができます．〈番号〉が省略された場合には0と解釈されます．

〈開始番地〉は〈番号〉によって指定されるユーザー関数の実行開始番地を直前に実行されたDEF SEG文で指定されたセグメントベースからの相対アドレスで指定します．

**参照**：DEF SEG, CLEAR, USR, BLOAD, BSAVE

**サンプルプログラム**

```
100 CLEAR ,&H3E00:DEF SEG=&H3E00:BLOAD"SAMPLE"
110 DEF USR=&H100
120 DEF USR1=&H200
130 PRINT USR(100)
140 A=USR1(1.23)
```

# DELETE

機　能　**プログラムの部分削除を行います.**

書　式　DELETE〔〈**始点行番号**〉〕〔— 〈**終点行番号**〉〕

文　例　DELETE 10-230

解　説　〈始点行番号〉から〈終点行番号〉までのプログラムを削除します.〈始点行番号〉だけを指定した場合はその行だけを削除し，マイナス記号以下〈終点行番号〉を指定した場合には，プログラムの先頭から指定行までを削除します.

# DIM

| | |
|---|---|
| 機　能 | **配列変数の要素の大きさを指定し，メモリ領域に割り当てます．** |
| 書　式 | DIM 〈変数名〉(〈添字の最大値〉〔,〈添字の最大値〉…〕) |
| 文　例 | DIM A(12, 2) |

**解　説**

配列変数の添字の最大値を設定し，同時にメモリ上にその配列の領域を割り当てます．DIM 文で宣言せずに配列変数を用いた場合，その添字の最大値は10とみなされます．もし，設定された最大値より大きい値の添字が用いられた場合は，〝Subscript out of range〞エラーが起こります．

添字の最小値は，OPTION BASE 文によって，0か1に指定することができます．

また，DIM 文が実行された時点で，その配列のすべての要素の値は0（文字型の場合はヌルストリング）に設定されます．

なお，数値配列の場合，1個の配列に対し指定できる要素の数は次のとおりです．

整数型の場合：32767個

単精度実数型の場合：16383個

倍精度実数型の場合：8191個

参照：OPTION BASE, ERASE

**サンプルプログラム**

```
100 '九九の表を作る
110 OPTION BASE 1
120 DIM KUKU(9,9)
130 FOR I=1 TO 9:FOR J=1 TO 9
140   KUKU(I,J)=I*J
150 NEXT J,I
160 FOR I=1 TO 9:FOR J=1 TO 9
170   PRINT USING "####";KUKU(J,I);
180 NEXT J:PRINT
190 NEXT I
200 END

run
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
   4   8  12  16  20  24  28  32  36
   5  10  15  20  25  30  35  40  45
   6  12  18  24  30  36  42  48  54
   7  14  21  28  35  42  49  56  63
   8  16  24  32  40  48  56  64  72
   9  18  27  36  45  54  63  72  81
Ok
```

# DRAW　(DISK モード)

**機　能**　**グラフィック描画サブコマンド列に従って，ワールド座標上で図形を描きます．**

**書　式**　DRAW 〈文字列式〉

**文　例**　DRAW "U50R50D50L50"

**解　説**　文字列式内に指定可能なグラフィック描画サブコマンドは，一般的には次のような形式です．

**〔〈修飾子〉〕〈識別子〉〔〈パラメータ〉〕〔,〈パラメータ〉〕…**

〈修飾子〉としては，BまたはNを指定することができ，次のような意味をもちます．

B　〈識別子〉で示されるサブコマンドの実行は行わず，LPのみをサブコマンド実行後の値とします．

N　〈識別子〉で示されるサブコマンド実行後，LPを(0，0)とします．

BおよびNを同時に指定した場合には，〈識別子〉で示されるサブコマンドの実行は行われず，LPが(0，0)となります．

〈識別子〉は，いろいろな描画機能を表わす英字1文字のサブコマンドを指定するものです．

〈パラメータ〉は，サブコマンドに対応した数値定数または変数で，以下の形式で指定します．

**〈数値定数〉**〔；〕または＝ **〈変数〉**；

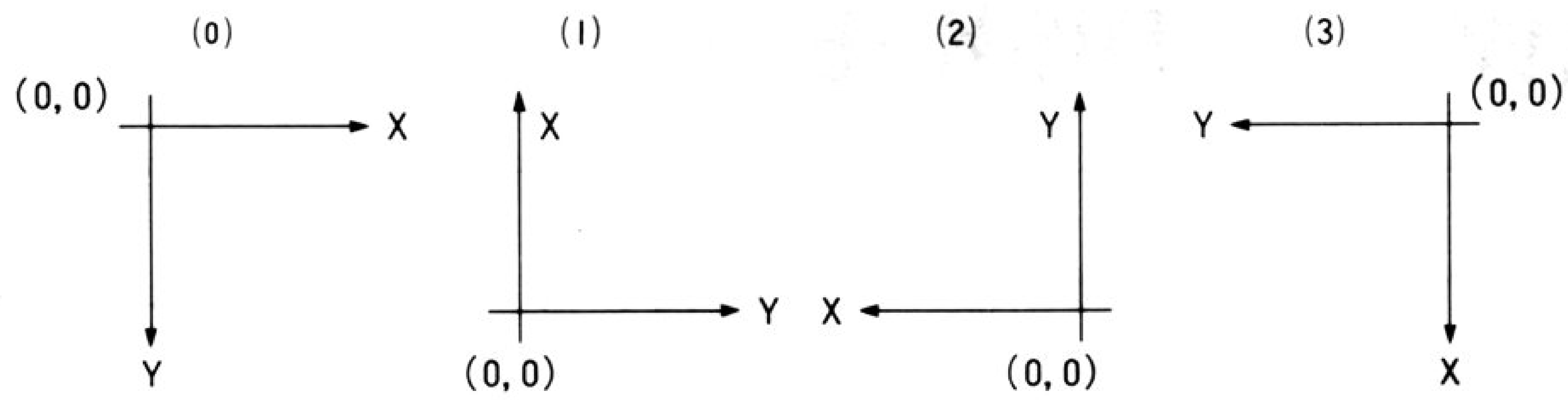
数値定数として指数形式を用いることはできません．また，16進表記で数値定数を表す場合には，その後に必ず；(セミコロン)を付けなければなりません．

〈識別子〉として指定可能なサブコマンドとその〈パラメータ〉を以下に示します．

A〈パラメータ1〉

グラフィック描画サブコマンドの現座標系を規定します．〈パラメータ1〉は，0～3までの整数値で，それぞれ以下の座標系を意味します．

Aサブコマンドによって規定された座標系はDRAW文のグラフィック描画サブコマンドに対してのみ有効なもので，ワールド座標系が変化するわけではありません．初期状態は0となっています．




C 〈パラメータ 1〉

グラフィック描画サブコマンドによって描画される図形のパレット番号を規定します．〈パラメータ 1〉はパレット番号を指定します．Cサブコマンドによって規定されたパレット番号は，グラフィック描画サブコマンドに対してのみ有効なものであり，フォアグラウンドカラーを変更するものではありません．

初期状態は，フォアグラウンドカラーとなっています．

D 〈パラメータ 1〉

LPから，現座標系のY軸に沿って〈パラメータ 1〉×スケール値分正の方向へ移動した点まで直線を描画します．

E 〈パラメータ 1〉

LPから，現座標系のX軸に沿って〈パラメータ 1〉×スケール値分正の方向に，さらにY軸に沿って〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します．

F 〈パラメータ 1〉

LPから，現座標系のX軸に沿って〈パラメータ 1〉×スケール値分負の方向に，さらにY軸に沿って〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します．

G 〈パラメータ 1〉

LPから，現座標系のX軸に沿って〈パラメータ 1〉×スケール値分負の方向に，さらにY軸に沿って〈パラメータ 1〉×スケール値分正の方向に移動した点まで直線を描画します．

H 〈パラメータ 1〉

LPから，現座標系のX軸に沿って〈パラメータ 1〉×スケール値分正の方向に，さらにY軸に沿って〈パラメータ 1〉×スケール値分正の方向に移動した点まで直線を描画します．

L 〈パラメータ 1〉

LPから，現座標系のX軸に沿って〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します．

**M 〈パラメータ1〉，〈パラメータ2〉**

LPから，ワールド座標系のX座標＝〈パラメータ1〉，Y座標＝〈パラメータ2〉の点まで直線を描画します．

**P〔〈パラメータ1〉〕**

LPを含み，〈パラメータ1〉のパレット番号で描かれた境界線で囲まれた領域をぬりつぶします．

〈パラメータ1〉が省略された場合には，Cサブコマンドで指定されているパレット番号〈パラメータ1〉の指定とみなします．Pサブコマンドを実行するときは，LPはウィンドウ内になければなりません．

Pサブコマンドによりぬりつぶされる色などの指定方法については，Zサブコマンドを参照してください．

**Q 〈パラメータ1〉，〈パラメータ2〉**

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分，Y軸に沿って〈パラメータ2〉×スケール値分それぞれ正の方向へ移動した点までを対角線とする長方形を描画します．

**R 〈パラメータ1〉**

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分正の方向に移動した点まで直線を描画します．

**S 〈パラメータ1〉**

グラフィック描画サブコマンドで指定される相対位置のパラメータのスケール値（倍率）を規定します．

〈パラメータ1〉はスケール値を表わし，0より大きな値でなければなりません．

初期状態は，スケール値＝1となっています．

**T 〈パラメータ1〉，〈パラメータ2〉**

Qサブコマンドと同様に，長方形を描画すると同時に，その内部をぬりつぶします．

ぬりつぶされる色などの指定方法については，Zサブコマンドを参照してください．

**U 〈パラメータ1〉**

LPから，現座標系のY軸に沿って，〈パラメータ1〉×スケール値分負の方向に移動した点まで直線を描画します．

**W 〈パラメータ1〉，〈パラメータ2〉**

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分，Y軸に沿って〈パラメータ2〉×スケール値分それぞれ正の方向へ移動した点ま

で直線を描画します.

**X 〈パラメータ 1〉**

〈パラメータ 1〉で指定される文字変数内のグラフィック描画サブコマンド列に従い図形を描画します.

文字列変数内に，再びXサブコマンドを含めることも可能です.

〈パラメータ 1〉 として文字列変数以外を指定することはできません.

**Y 〈パラメータ 1〉**

D, E, F, G, H, L, M, Q, R, UおよびWサブコマンドで描画する直線または長方形のラインスタイルを規定します. 〈パラメータ 1〉は, &H0～&HFFFF の範囲の整数でなければなりません.

ラインスタイルの内容については, LINE を参照してください. 初期状態は, ラインスタイル＝&HFFFF となっています.

**Z 〈パラメータ 1〉**

TおよびPサブコマンドで，ぬりつぶす際のぬりつぶしの色または〈タイルストリング〉による模様を規定します. 〈パラメータ 1〉は，パレット番号または〈タイルストリング〉が格納されている文字列変数でなければなりません.

〈タイルストリング〉については,〔2〕PAINT を参照してください. 初期状態は, Cサブコマンドにより規定されるパレット番号（Cサブコマンド未実行時は，フォアグラウンドカラー）となっています.

D, E, F, G, H, L, M, Q, R, S, T, UおよびWサブコマンドの実行では, 修飾子としてNが指定されていない限り, LP は指定された点に移ります. A, C, P, S, YおよびZサブコマンドの実行では LP は変化しません.

また, A, C, S, YおよびZサブコマンドの指定は, 指定のある DRAW 文だけでなく, 以降の DRAW 文にもその指定が引き継がれます. これらの指定は, SCREEN 文の実行時に初期状態に戻ります.

参照：LINE,〔2〕PAINT

サンプルプログラム

```
100 SCREEN 0,0,0,1
110 POINT(320,100)
120 A$="U60R60D60L60"
130 DRAW A$
140 DRAW "BE45A2S0.5X=A$;"
150 DRAW "A0BE10P"
160 END
```

# DSKO$　(DISK モード)

**機能**　ディスクに対して直接書き込みを行います.

**書式**　DSKO$ 〈ドライブ番号〉, 〈ヘッド番号〉, 〈トラック番号〉, 〈セクタ番号〉

**文例**　DSKO$ 1, 0, 19, 1

**解説**

通常のファイル操作とは無関係に，ディスク上の指定したセクタに直接書き込みを行います．この文は，既存のディスクファイルを壊す恐れがありますので，ディスク及びファイルの構成を正しく理解した上で用いてください．これらについては，ユーザーズマニュアルで説明されています．

DSKO$は，そのパラメータにより指定されたセクタに，システム用バッファの内容256バイトを書き込みます．システム用バッファは，通常のディスクファイル操作の際には用いることはできませんが，DSKI$とDSKO$による直接操作の際にのみ，ユーザーが用いることができます．

書き込むデータの準備は，ランダムアクセスの場合と同じように FIELD 文により，システム用バッファへ変数領域を割り付けた後, LSET, RSET により行います．あるいは，VARPTR 関数により，システム用バッファの置かれているメモリ番地を調べた上で，POKE 文によりバッファ中にデータを準備することもできます．

〈ヘッド番号〉，〈トラック番号〉，〈セクタ番号〉として指定できる値の範囲は，指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが，BASIC は，これらの値の範囲を自動的に調べ，不当であった場合には〝Bad track／sector〟エラーとします．詳しくはユーザーズマニュアルを参照してください．

**注意**：それぞれのディスクの諸元についてはDSKF 関数により調べることができます．詳しくはユーザーズマニュアルを参照してください．

**参照**：DSKI$，FIELD，DSKF，VARPTR

サンプルプログラム

```
100 '-------- WRITE TO THE DISK --------
110 ' このプログラムはＤＩＳＫに書き込まれます
120 '***** 不用意に使用しないでください ******
130  FIELD #0,128 AS A$,128 AS B$
140  LSET A$=STRING$(128,CHR$(&HFF))
150  LSET B$=STRING$(128,CHR$(&HFF))
160 *START
170  INPUT "Drive";DR
180  IF DR<1 OR DR>8 THEN *START
190  INPUT "Surface";SU
200 *REENT
210  INPUT "Track,Secter";TR,SE
220  IF TR<0 OR TR>DSKF(DR,0) THEN *REENT
230  IF SE<1 OR SE>DSKF(DR,1) THEN *REENT
240  DSKO$ DR,SU,TR,SE
250  END
```

# EDIT　(DISK モード)

**機能**　**指定された行を画面上へ表示し，以降 [ROLL UP]，[ROLL DOWN] キー等によるプログラム編集を可能にします．**

**書式**　EDIT ｜〈行番号〉｜
　　　　　　 ｜　　．　　｜

**文例**　EDIT 30

**解説**　画面上での編集ができるように，指定された行を表示し，カーソルをその行の先頭に置きます．以降，[ROLL UP]，[ROLL DOWN] キーを始めとする各種プログラム編集キーを使用し，プログラムの編集が可能となります．

Pオプション付きでセーブされたファイルをロードしてEDITコマンドにより，内容を表示，変更しようとした場合には，〝Illegal function call〟のエラーとなります．

E

# END

**機 能**　**プログラムの終了を宣言します.**

**書 式**　END

**文 例**　END

**解 説**　プログラムの実行を終了し，すべてのファイルをクローズしてコマンドレベルに戻ります.

END 文はプログラムの実行を終了させたい所に置けばよく，また，いくつ置いてもかまいません. プログラムの最後の END 文は省略することができますが，この場合，ファイルのクローズは行われません.

**サンプルプログラム**

```
100 WIDTH 80,25
110 COLOR 7
120 CLS
130 *LOOP
140  PRINT TIME$
150  A$=INKEY$
160   IF A$="s" THEN END
170 GOTO *LOOP

run
18:04:15
18:04:15
18:04:15
18:04:15
18:04:15
18:04:16
18:04:16
18:04:16
18:04:16
18:04:16
18:04:16
18:04:16
Ok
```

# ERASE

機 能　**配列変数を消去します.**

書 式　ERASE 〈配列変数名〉〔,〈配列変数名〉…〕

文 例　ERASE C, D

解 説　指定した配列変数を消去します. 以後, 同一変数名の配列変数を DIM 文で宣言することができます. もし ERASE 文を実行せずに同一変数名で宣言すると, "Duplicate Definition" エラーが起こります.

また, その配列変数に割り当てられていたメモリエリアを, 他の目的に使用することもできます.

参照：DIM

# ERROR

| | |
|---|---|
| 機能 | **エラー発生のシミュレート，エラー番号のユーザー定義を行います．** |
| 書式 | ERROR <整数表記> |
| 文例 | ERROR 200 |

**解説**

<整数表記>の値は0から255までの範囲の数を指定します．この値が，すでにBASICでエラーコードとして使われている場合，ERROR文は，そのエラーの発生をシミュレートします．

また，未定義のエラーコードのエラーをプログラム中で条件に応じてERROR文で発生させ，ON ERROR GOTO文によって指定されたエラー処理ルーチン内でエラー処理を行うことができます．

**参照**：ON ERROR GOTO，ERL／ERR，エラーコード表

サンプルプログラム

```
100  ON ERROR GOTO *ERMESSAGE
110 *NYURYOKU
120  INPUT "３桁の整数入力せよ．",N
130   IF FIX(N)<>N THEN ERROR 251
140   IF N<100 OR N>999 THEN ERROR 250
150  PRINT USING "<###>";N
160 END
170 *ERMESSAGE
180  IF ERR=250 THEN PRINT "３桁とは１００から９９９！！！"
190  IF ERR=251 THEN PRINT "実数は入力できません．"
200 PRINT
210 RESUME *NYURYOKU


run
３桁の整数入力せよ．12457
３桁とは１００から９９９！！！

３桁の整数入力せよ．12.336
実数は入力できません．

３桁の整数入力せよ．777
<777>
Ok
```

# FIELD



**機能**　**ランダムファイルバッファに変数領域を割り当てます．**

**書式**　FIELD 〔#〕〈ファイル番号〉, 〈フィールド幅〉 AS 〈文字変数〉〔, 〈フィールド幅〉 AS 〈文字変数〉 …〕

**文例**　FIELD #1, 128 AS A$, 64 AS B$, 64 AS C$

**解説**

ファイルバッファをランダムファイル バッファとして使用するために，ファイルバッファの中に文字変数の領域を割り当てます．この変数を引用することにより，バッファのデータの設定，バッファに設定されているデータの参照が可能になります．

PUT 文，GET 文を使ってランダムファイルのアクセスを行う前に必ず，FIELD 文が実行されていなければなりません．

〈ファイル番号〉は，OPEN 文によってファイルをオープンしたときに指定した番号です．

〈フィールド幅〉は〈文字変数〉に割り当てる文字数です．1つのバッファには文字数の合計が，256文字までならば，いくつの文字変数を割り当ててもかまいません．

また，1つのバッファに対して，異なった形式の FIELD 文を実行してもかまいません(サンプルプログラムを参照してください)．

各フィールドへの値の代入を通常の代入文あるいは INPUT 文等で行わないでください．もし行うと，その変数は，一般の文字変数領域に登録され，FIELD 文でのバッファの割り当てが無効となり，正しいファイルの入出力が行われなくなります．

また，日本語をフィールドに設定する場合，日本語1文字が2文字分のフィールドを必要とすること，および日本語文字列の前後にシフトコード(KI コードおよび KO コードそれぞれ2文字分)が挿入されることに注意してください．例えば，

LSET A$= "日本語"

のためには最低10文字分のフィールドを必要とします．したがって，10 AS A$としておく必要があります．

各フィールドへの値の代入は LSET あるいは RSET 文で行います．フィールド幅を越えて値を代入することはできません．数値をフィールドに設定する場合はその数値をいったん文字型に変換してから代入します（MKI$, MKS$,

MKD$を使用します).

参照：OPEN, GET, PUT, LSET／RSET, MKI$, MKS$, MKD$, CVI, CVS, CVD

サンプルプログラム

```
100 OPEN "RTEST.DAT" AS #1
110 FIELD #1,30 AS A$,30 AS B$,30 AS C$
120 '-- DUMMY$ は A$+B$+C$ のためのダミーです --
130 FIELD #1,90 AS DUMMY$,2 AS D$,4 AS E$,8 AS F$
140 LSET A$="NEC"
150 LSET B$="Personal"
160 LSET C$="Computer"
170 RSET D$=MKI$(12345)
180 RSET E$=MKS$(228000!)
190 RSET F$=MKD$(6.626180000000018D-34)
200 PUT #1,1
210 GET #1,1
220 PRINT A$
230 PRINT B$
240 PRINT C$
250 D%=CVI(D$)
260 E=CVS(E$)
270 F#=CVD(F$)
280 PRINT D%:PRINT E:PRINT F#
290 END
```

```
run
NEC
Personal
Computer
 12345
 228000
 6.626180000000018D-34
Ok
```

# FILES／LFILES　（DISK モード）

**機　能**　ディスクに入っているファイルの名前，種類と大きさを出力します．

**書　式**　1） FILES〔〈ドライブ番号〉〕

2） LFILES〔〈ドライブ番号〉〕

**文　例**　FILES 2

**解　説**　〈ドライブ番号〉で指定された，ディスク上に登録されているファイルの名前，種類と，その大きさを表示します．書式1）の場合ディスプレイに，書式2）の場合プリンタに出力します．ファイルの大きさは，クラスタという単位で表示されます．クラスタについては，ユーザーズマニュアルを参照してください．

〈ドライブ番号〉が省略された場合には，ドライブ1が選択されます．

ファイルの種類は，出力されるファイルの名前と，タイプとの区切り文字によって表されます．区切り記号が空白の場合には，そのファイルはアスキーセーブされたプログラムファイル，あるいは，OPEN 文によって作られた，BASIC のデータファイルであることを示します．区切り文字がピリオドの場合には，バイナリセーブによって作られたプログラムファイル，アスタリスクの場合には，BSAVE によって作られた機械語ファイルであることを表します．

参照：SAVE，BASVE

サンプルプログラム

```
files
menu   .    2    BUNSET SU   73    KANJI  DIC 5    format.nip 3    backup.n88 3
setinf.n88 1    xfiles.n88 1    sysgen.nip 2    format.hd  2    recov .hd  3
xfiles.hd  1    dir   .hd  1    backup.hd  2    mkfont.n88 2    switch.n88 4
dicmen.n88 6    DDconv.n88 1    mouse *cod 1    tele  .n88 5    XMODEM*BIN 2
Ok
```

F

# FOR…TO…STEP〜NEXT

**機能**　FOR 文から NEXT 文までの一連の命令を指定回数だけ繰り返して実行します.

**書式**

```
FOR 〈変数名〉= 〈初期値〉 TO 〈終値〉〔STEP 〈増分〉〕
 〜
NEXT 〔〔〈変数名〉〕〔, 〈変数名〉〕〕
```

**文例**

```
FOR J=0 TO 100 STEP 2
 〜
NEXT J
```

**解説**

〈変数〉はカウンタとして使われ，最初に初期値に設定されます．そして，FOR 文以降から NEXT 文までが実行され，カウンタの値は STEP によって指定された〈増分〉だけ増減されます．次にカウンタの値が〈終値〉と比べられ，カウンタの値が〈終値〉に達していなければ，FOR 文の次の文へ戻り，同じ処理が繰り返されます．

STEP 文が省略された場合，〈増分〉は＋1とみなされます．また，〈増分〉は，負の値をとることもできます．

次の場合には，FOR〜NEXT 文は実行されずに，NEXT 文の次へ実行が移ります．

1)〈増分〉が正の値で，〈初期値〉が〈終値〉より大きい場合．

2)〈増分〉が負の値で，〈初期値〉が〈終値〉より小さい場合．

ただし，〈変数〉には〈初期値〉が代入されます．

FOR〜NEXT は入れ子構造にすることができます．これは1つの FOR〜NEXT のなかにもう一つの FOR〜NEXT を置くことができるということで，この場合，それぞれの〈変数〉は別のものを使わなければなりません．また，1つの FOR〜NEXT は完全に他の FOR〜NEXT の内部になければなりません．

FOR 文と NEXT 文とは必ず1対1に対応していなければなりません．したがって，IF 文中に NEXT 文を置く事ができない場合があります．

**サンプルプログラム**

```
100 SCREEN 0:CLS 3
110 FOR R=10 TO 200 STEP 20
120  FOR I=0 TO 3.14 STEP .05
130   Y=SIN(I)*COS(I)
140   X=SIN(I)*SIN(I)
150   Y=Y*R+100
160   X=X*R*2.5+100-C
170   PSET (X,Y)
180  NEXT I
190 C=C+10
200 NEXT R
210 END
```

# GET

**機 能**　**ファイル中のデータをバッファに読み込みます.**

**書 式**　GET 〔#〕<ファイル番号>〔, <数>〕

**文 例**　GET 3, 5

**解 説**

〈ファイル番号〉で指定されたファイル中のデータを，対応するバッファ中に読み込みます. GET は，指定されたファイルがディスクファイルかどうかで，その動作が異なります.

〈ファイル番号〉で指定されたファイルがディスクファイルである場合には，GET はランダムファイルからのレコードを読み込みます. 指定されたファイルはランダムアクセスのモードでオープンされていなければなりません.〈数〉はファイルのレコード番号として解釈され，指定されたレコードがバッファに読み込まれます. レコード番号が省略された場合には，直前に行われた，GET，PUT で指定されたレコードの次のレコードが読み込まれます. レコード番号の最小値は 1，最大値は65000です.

GET を行った後で，INPUT #，LINE INPUT #等のシーケンシャル・アクセス入力を行った場合には，バッファに入っているデータから入力が行われますので注意してください.

〈ファイル番号〉で指定されたファイルがキーボードファイルの場合には，GET は，指定されたファイルに対応する装置から指定した文字数をバッファ中に読み込みます. 指定されたファイルは入力モードでオープンされていなければなりません.〈数〉は，ファイルからバッファに読み込む文字数を意味し，0 から255までの値で指定します.〈数〉が省略された時，及び 0 が指定された時には256文字を読み込みます.

GET はバッファへの読み込みを行うものです. 読み込んだデータはフィールド変数で参照することができます.

参照：OPEN, FIELD, PUT

サンプルプログラム

```
100 OPEN "DATA" AS #1
110 FIELD #1,4 AS I$,20 AS N$,50 AS A$
120 INPUT "会員コード";I
130 IF I=0 THEN END
140 GET #1,I
150 PRINT "会員コード";CVS(I$);
160 PRINT "氏名";N$;
170 PRINT "住所";A$
180 GOTO 120
```

# GET@

---

**機　能**　**画面上のグラフィックパターンを配列変数に読み込みます.**

**書　式**　GET〔@〕(Sx1, Sy1)－ | (Sx2, Sy2) / STEP(x, y) | , **〈配列変数名〉〔(〈要素ナンバー〉)〕**

**文　例**　GET (100, 50)－ (130, 70), G%

**解　説**

画面上のスクリーン座標の2点(Sx1, Sy1)と(Sx2, Sy2)を結ぶ線を対角とする四角形の領域を,〈配列変数名〉で指定された配列変数に読み込みます.

気を付けなければならないのは, BASICのグラフィック命令のほとんどがワールド座標を用いて使用するのに対し, このGET@命令とPUT@命令(PUT@参照)は, ビューポート内に展開されるスクリーン座標を使用しなければならないことです.また2つ目の座標指定には相対座標による指定が許されています.

〈配列変数名〉は, ユーザーがグラフィックデータを読み込むために用意する数値型の配列変数の名前です. またこの配列変数に対して〈要素ナンバー〉を指定することもできます. この〈要素ナンバー〉は, グラフィックパターンを配列変数に読み込む際に, どの要素から格納し始めるかの指定です. 省略した場合は配列の最初から格納します. 1つの配列変数に対して複数のグラフィックデータを格納する場合に使うことができます.

確保すべき配列変数の大きさは, 読み込む領域の大きさ, 現在のスクリーンモード, 配列変数の型によって異なりますが, 1つの配列に1つのパターンしか読み込まない場合は, 次の計算式で求めることができます. 複数パターンの場合はそれぞれのパターンについて計算し, その分確保しなければなりません.

**〈必要なバイト数〉=((〈横のドット数〉+7)¥8)*〈縦のドット数〉*M+4**

ただし　白黒モードの時 —— M=1
8色カラーモードの時 —— M=3
16色カラーモードの時 —— M=4

**〈添字の値〉=〈必要なバイト数〉¥N+1**

Nは配列変数の型によって変わります.

整数型配列 —— N=2
単精度型配列 —— N=4
倍精度型配列 —— N=8

ただしこの〈添字の値〉とは, 1次元配列で添字の底が0の時(OPTION BASE参照)の最小値です. 一般にGET@では1次元の配列変数を用います.

このGET＠文は後述するPUT＠文と密接な関係にあり，通常ペアで使用されます．

GET＠命令実行後，LP(Last referenced point)は($Sx_2$，$Sy_2$)に移されます．

参照：PUT＠，OPTION BASE

サンプルプログラム

```
100 SCREEN 0,0:CLS 3
110 XD=40:YD=20
120 BYTE=((XD+7)¥8)*YD*3+4
130 FACT=BYTE¥2+1
140 DIM G%(FACT)
150 '
160 LINE(0,0)-STEP(XD-1,YD-1),1,B
170 CIRCLE(XD/2-1,YD/2-1),YD/2,4
180 GET(0,0)-STEP(XD-1,YD-1),G%
190 '
200 FOR X=0 TO 500 STEP 100
210 PUT (X,100),G%:NEXT
```

# GOSUB～RETURN

機　能　サブルーチンの呼び出し，およびサブルーチンからのもどりを制御します．

書　式　GOSUB 〈行番号〉
　　　　　∫
　　　　RETURN〔〈行番号〉〕

文　例　GOSUB ＊SUB1

解　説

サブルーチンとは，他から独立したプログラムで，RETURN 文で終っているものをいいます．これらは，GOSUB 文によって呼び出されます．RETURN 文は対応する GOSUB 文の次の文に制御をもどします．１つのサブルーチンの中から他のサブルーチンをコールすることもでき，これをサブルーチンの多重化と呼びます．この多重化はメモリのスタック領域の容量が許す限り行う事ができ，もし，スタック領域が足りなくなった場合には，〝Out of memory〃エラーが起こります．

サブルーチンは必ず GOSUB 文によってコールされなければなりません．もし，単独で実行した場合，RETURN 文に出会うと，〝RETURN without GOSUB〃エラーが起こります．

また，１つのサブルーチン内に複数の RETURN 文があってもかまいませんが，正しく GOSUB 文と対応していなければなりません．

なお，.RETURN 文では〈行番号〉を指定して，条件に応じて特定の行へ強制的にリターンさせることができます．

参照：CLEAR

サンプルプログラム

```
100 '三角形の面積を求めよ.
110 *START
120  INPUT "底辺：";TEIHEN
130  INPUT "高さ：";TAKASA
140  GOSUB *MENSEKI
150  PRINT "面積は";MENSEKI
160  PRINT
170 GOTO *START
180 *MENSEKI
190  MENSEKI=TEIHEN*TAKASA/2
200 RETURN
```

```
run
底辺：? 100
高さ：? 50
面積は 2500

底辺：? 145
高さ：? 20
面積は 1450

底辺：? 10000
高さ：? 34
面積は 170000

底辺：? 356
高さ：? 180
面積は 32040

底辺：?
Break in 120
Ok
```

# GOTO／GO TO

| | |
|---|---|
| 機　能 | 指定された行へ制御を移行します. |
| 書　式 | 1) GOTO <行番号><br>2) GO TO <行番号> |
| 文　例 | GOTO 500<br>GOTO ＊S1 |
| 解　説 | <行番号> の行へ制御を移行します.<br>書式1), 書式2) は全く同機能です. |

**注意**： 〝GO〞 と 〝TO〞 の間のスペースは１個のみが許されます. ２個以上のスペースを入れた場合, BASIC はそれを GOTO 文とは解釈しません.

サンプルプログラム

```
100 *START
110  PRINT "How do you do?"
120  GOTO *START


run
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
^C
Break in 110
Ok
```

# HELP ON／OFF／STOP

機　能　ヘルプキーによる割り込みの許可，禁止，停止を制御します．

書　式　1）HELP ON

2）HELP OFF

3）HELP STOP

文　例　HELP OFF

解　説　書式1）割り込み動作を許可します．
この命令実行後，ヘルプキーを押すごとに割り込みが発生し，ON HELP GOSUB によって定義された処理ルーチンに分岐します．

書式2）割り込み動作を禁止します．この命令実行後，ヘルプキーを押しても処理ルーチンへの分岐は起こりません（ヘルプキーは通常の動作をします）．

書式3）割り込み動作を停止します．この命令実行後，ヘルプキーを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後 HELP ON 命令によって割り込みが許可されると，先程のヘルプキーを押したことによって処理ルーチンに分岐します．

注意：プログラム終了時には HELP OFF を実行しておいてください．さもないと本来のヘルプキーの機能が失われます．

参照：ON HELP GOSUB, KEY ON／OFF／STOP

サンプルプログラム

```
100 ON HELP GOSUB *MENU
110 HELP ON
     .
     .
     .
1000 *MENU
     .
     .
     .
1200 RETURN
```

# IF…THEN…ELSE/IF…GOTO…ELSE

**機　能**　論理式の条件判断を行います.

**書　式**

```
IF 〈論理式〉 | THEN | 〈文〉    | | 〔ELSE | 〈文〉    | 〕
              |      | 〈行番号〉| |        | 〈行番号〉|
              | GOTO 〈行番号〉    |
```

**文　例**　IF A$= "y" THEN ＊START ELSE 120

**解　説**　論理式の条件によってプログラムの実行を制御します. 即ち, 〈論理式〉が真(0以外) ならば, THEN あるいは GOTO 文以降を実行し, 偽(0)ならば ELSE 文以降が実行されます. もし, ELSE 文以降が省略されている場合には, 次の行が実行されます.

IF…THEN…ELSE 文において, THEN 文は ELSE に続けて別の IF 文を置いて多重にすることができます. ただし, 多重できるのは1行に書ける範囲に限られます.

**サンプルプログラム**

```
100 '-- real or imaginary --
110 INPUT A
120 FLG=0
130  IF A<0 THEN A=ABS(A):FLG=1
140 PRINT SQR(A);
150  IF FLG THEN PRINT "i" ELSE PRINT
160 END


run
? 743
 27.258
Ok
run
? -653.23
 25.5584 i
Ok
run
? -3.9576
 1.98937 i
Ok
```

# INPUT

**機　能**　キーボードからデータを入力し，変数に代入します．

**書　式**　INPUT〔〈プロンプト文〉 { ; | , } 〕〈変数〉〔,〈変数〉…〕

**文　例**　INPUT "住所"；A$

**解　説**

INPUT 文が実行されると，プログラムはキーボードからの入力待ちの状態となります．プロンプト文はデータの入力を受ける前に画面に表示されるメッセージです．〈プロンプト文〉にセミコロン（；）が続いた場合にはさらにクエッションマーク（？）と1桁のスペースが画面に表示されますが，コンマ（,）が続いた場合には〈プロンプト文〉の後には何も出力されません．

データはリターンキーを押すことによって入力され，変数に代入されます．〈変数〉をコンマ（,）で区切って複数個指定した場合には，入力するデータもコンマで区切って〈変数〉の数だけ入力しなければなりません．また，対応する〈変数〉とデータの型とは一致していなければなりません．もし，個数や型が違っていた場合には "?Redo from start" を出力して再び入力待ちとなります．

また，何もせずにリターンキーを押した場合には0やヌルストリングが入力されたとみなされます．この場合も〈変数〉が複数の場合，必要数のコンマだけは入力しなければなりません．

文字変数に文字列を代入する場合，コンマや文字列の前後の意味のある空白を入力したいときは，ダブルクォーテーション(")で文字列を囲む必要があります．この場合は文字としてダブルクォーテーションを入力することはできません．これ以外の場合には，文字列をダブルクォーテーションで囲む必要はありません．

入力を中断したい場合，STOP キーあるいは CTRL-C キーを入力します．プログラムの実行中に入力を中断すると，STOP ON および STOP STOP の状態にない限り，プログラムの実行も中断され，コマンドモードにもどります．また，それまでに入力した値はすべて無効となります（変数には値が代入されません）．

日本語の入力を行う場合，CTRL-XFER を入力することにより日本語入力モードにします．

日本語入力モードにおける日本語の入力方法に関してはユーザーズマニュアルを参照してください．

**参照**：LINE INPUT，KINPUT，INPUT WAIT

サンプル
プログラム

```
100 INPUT "名前  ";NA$
110 INPUT "性別  ";SEX$
120 INPUT "年齢  ";OLD
130 PRINT "名前  : ",NA$
140 PRINT "性別  : ",SEX$
150 PRINT "年齢  : ",AKCNV$(STR$(OLD));"才"
```

```
run
名前  ? 松沢 謙治
性別  ? 男
年齢  ? 23
名前  :       松沢 謙治
性別  :       男
年齢  :         ２３才
Ok
```

# INPUT #

**機能**　シーケンシャルファイルからのデータを読み込みます.

**書式**　INPUT # 〈ファイル番号〉, 〈変数〉〔, 〈変数〉…〕

**文例**　INPUT #1, A, B

**解説**

データを読み込む対象がシーケンシャルファイルであることと，クエッションマーク（?）が出力されないことを除けば，INPUT 文と同じです.

〈ファイル番号〉は，OPEN 文によって，そのファイルを入力としてオープンしたときに使った番号です.

ファイルに書き込まれているデータは，INPUT 文に対して必要なデータと同様の正しい形になっていなければなりません.

すなわち，数値変数に代入される値（数字の並び）は空白，コンマあるいはキャリッジリターン（CHR $(13)）で，また，文字変数に代入される値（任意の文字の並び）はコンマあるいはキャリッジリターン（CHR $(13)）でそれぞれ区切られていなければなりません.

参照：OPEN

**サンプルプログラム**

```
100 '-- Sequential data PRINT#/INPUT# --
110 '
120  OPEN "STEST.DAT" FOR OUTPUT AS #1
130  PRINT #1,DATE$;",";TIME$
140 *DATAWRITE
150  INPUT "品名";NA$
160  IF NA$="END" OR NA$="end" THEN *EXIT1
170  INPUT "価格";PLC
180  INPUT "個数";N%
190  PRINT #1,NA$;",";PLC;N%
200  GOTO *DATAWRITE
210 *EXIT1
220  PRINT:CLOSE
230 '-- Sequential data read --
240  TOTAL=0
250  OPEN "STEST.DAT" FOR INPUT AS #1
260  INPUT #1,DA$,TI$
270  PRINT "日付  :  ";DA$,"時間  :  ";TI$
280  PRINT
290 *DATAREAD
300   IF EOF(1) THEN *EXIT2
310   INPUT #1,NA$,PLC,N%
320   SUM=PLC*N%
330   PRINT NA$;TAB(10);PLC;"*";N%;"=";SUM
340   TOTAL=TOTAL+SUM
350  GOTO *DATAREAD
360 *EXIT2
370  PRINT:PRINT "TOTAL=";TOTAL
380 END
```

```
run
品名? 米
価格? 2500
個数? 1
品名? 牛肉
価格? 800
個数? 4
品名? END

日付   :   85/06/06          時間   :   06:06:06

米           2500 * 1 = 2500
牛肉          800 * 4 = 3200

TOTAL= 5700
Ok
```

# INPUT WAIT

**機能**　キーボードからデータを入力し，変数に代入します．その際，入力待ち時間を制限することができます．

**書式**　INPUT WAIT 〈待ち時間〉, 〔〈プロンプト文〉 ｛ ; ｜ , ｝〕〈変数名〉〔, 〈変数名〉…〕

**文例**　INPUT WAIT 100, "Your name" ; NA$

**解説**

〈待ち時間〉で指定された時間だけキーボードからの入力を待ちます．〈待ち時間〉の単位は0.1秒です．

INPUT WAIT は待ち時間に制限があることを除けば，INPUT 文と同様に機能します．ただし，この文の後にマルチステートメントとして他の文を続けた場合，制限時間内に入力が行われた時のみ後の文を実行し，入力が行われなかった時は，後の文を無視し次の行へ実行が移りますから，プログラミングの際は注意してください．

**サンプルプログラム**

```
100 '-- computer teacher --
110 *ENTRY:A=INT(RND*100):B=INT(RND*100)
120  PRINT "Problem",
130  PRINT A;"+";B;"=";
140  INPUT WAIT 30,ANS:GOTO *EXIT
150  PRINT:PRINT "Time over !",:GOTO *ANS
160 *EXIT
170  IF ANS=A+B THEN PRINT "Right !":GOTO *ENTRY
180  PRINT "Wrong !",
190 *ANS:BEEP:PRINT "Answer =";A+B:GOTO *ENTRY
```

```
run
Problem        2 + 61 =? 63
Right !
Problem        96 + 97 =? 195
Wrong !       Answer = 193
Problem        38 + 84 =? 122
Right !
Problem        85 + 79 =?

Time over !   Answer = 164
Problem        92 + 36 =?
Break in 140
Ok
```

# KEY

| | |
|---|---|
| 機　能 | **キーボードの上部にあるプログラマブルファンクションキーに与える文字を定義します.** |
| 書　式 | KEY <キー番号>, <文字列> |
| 文　例 | KEY 1, "Yes"+CHR$(13)<br>KEY 3, CHR$(&H22)+"BASIC"+CHR$(&H22) |
| 解　説 | ファンクションキーは10個あり, 10個の<文字列>を記憶させておくことができます. 各ファンクションキーは最大15文字までの文字列およびコントロール文字をプログラムできます. キーボードから入力できない文字は, CHR$関数で与えます. |

**注意**：ファンクションキーに日本語文字列を定義することはできません.

# KEY LIST

| | |
|---|---|
| 機　能 | **ファンクションキーの内容を画面に表示します.** |
| 書　式 | KEY LIST |
| 文　例 | KEY LIST |

解　説　ファンクションキーの内容の一部は画面の最下行に表示させておくことができますが, KEY LIST では, そのすべての内容を画面に表示させることができます.

K

# KEY (n) ON／OFF／STOP

機能　ファンクションキーによる割り込みの許可，禁止，停止を制御します．

書式　1)　KEY〔(<キー番号>)〕ON

2)　KEY〔(<キー番号>)〕OFF

3)　KEY〔(<キー番号>)〕STOP

文例　KEY ON

KEY (3) ON

解説　ファンクションキーが押された時に起こる割り込みを許可するか (ON)，禁止するか (OFF)，停止するか (STOP) を宣言します．

〈キー番号〉とは，1から10までのプログラマブルファンクションキーの番号を表わします．省略された場合は1から10までのすべてのファンクションキーに対して実行します．

**書式1)** 割り込み動作を許可します．この命令実行後は，指定された〈キー番号〉のファンクションキーを押すごとに割り込みが発生し，ON KEY GOSUB によって定義された処理ルーチンに分岐します．

**書式2)** 割り込み動作を禁止します．この命令実行後は，ファンクションキーを押しても処理ルーチンへの分岐は起こりません(通常のファンクションキーの動作になります)．

**書式3)** 割り込み動作を停止します．この命令実行後は，ファンクションキーを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後 KEY(n)ON 命令によって割り込みが許可されると，先程のファンクションキーを押したことによって処理ルーチンに分岐します．

参照：ON KEY GOSUB

サンプルプログラム

```
100 ON KEY GOSUB *FKEY1,,*FKEY3
110 KEY(1) ON:KEY(3) ON
         .
         .
         .
1000 'FKEY1 PROCESSING
1010 *FKEY1
         .
         .
         .
1200 RETURN
2000 'FKEY3 PROCESSING
2010 *FKEY3
         .
         .
         .
2200 RETURN
```

K

# KILL　（DISK モード）

機 能　ディスクからファイルを削除します．

書 式　KILL 〈ファイルディスクリプタ〉

文 例　KILL "2 : DATA1"

解 説　〈ファイルディスクリプタ〉で指定したファイルをディスクから削除します．

削除するファイルはクローズ状態になっていなければなりません．もしオープン状態のファイルを削除しようとすると "File already open" エラーが起こります．また 1 つの KILL 命令では 1 つのファイルしか削除できません．

KILL 命令はすべてのディスクファイルについて使用できます．

# KINPUT　(DISK モード)

|機能| **キーボードから日本語を入力し文字変数に代入します.**

|書式| KINPUT <変数>

|文例| KINPUT A$

|解説| KINPUT 文が実行されると，自動的に日本語入力モードに入ります. INPUT 文との違いは自動的に日本語入力モードに入るか否かの違いです.

日本語入力モードにおける日本語の入力方法に関しては，ユーザーズマニュアルを参照してください.

KINPUT 文は，他の INPUT 関連文と異なり，1つの文では1つの変数しか指定できません. また，日本語文字以外の文字を入力することもできません.

**注意**：KINPUT 文での入力は偶数桁からになります. LOCATE 文で奇数桁を指定した場合は，その桁数に1を加えた桁が指定されたものと解釈します.

# KPLOAD　(DISK モード)

**機　能**　利用者定義文字パターンを登録します．

**書　式**　KPLOAD 〈漢字コード〉，〈整数型配列名〉

**文　例**　KPLOAD &H762A，CHRPTN%

**解　説**　〈漢字コード〉で指定された漢字の文字パターンとして，〈整数型配列名〉で指定されたものを登録します．

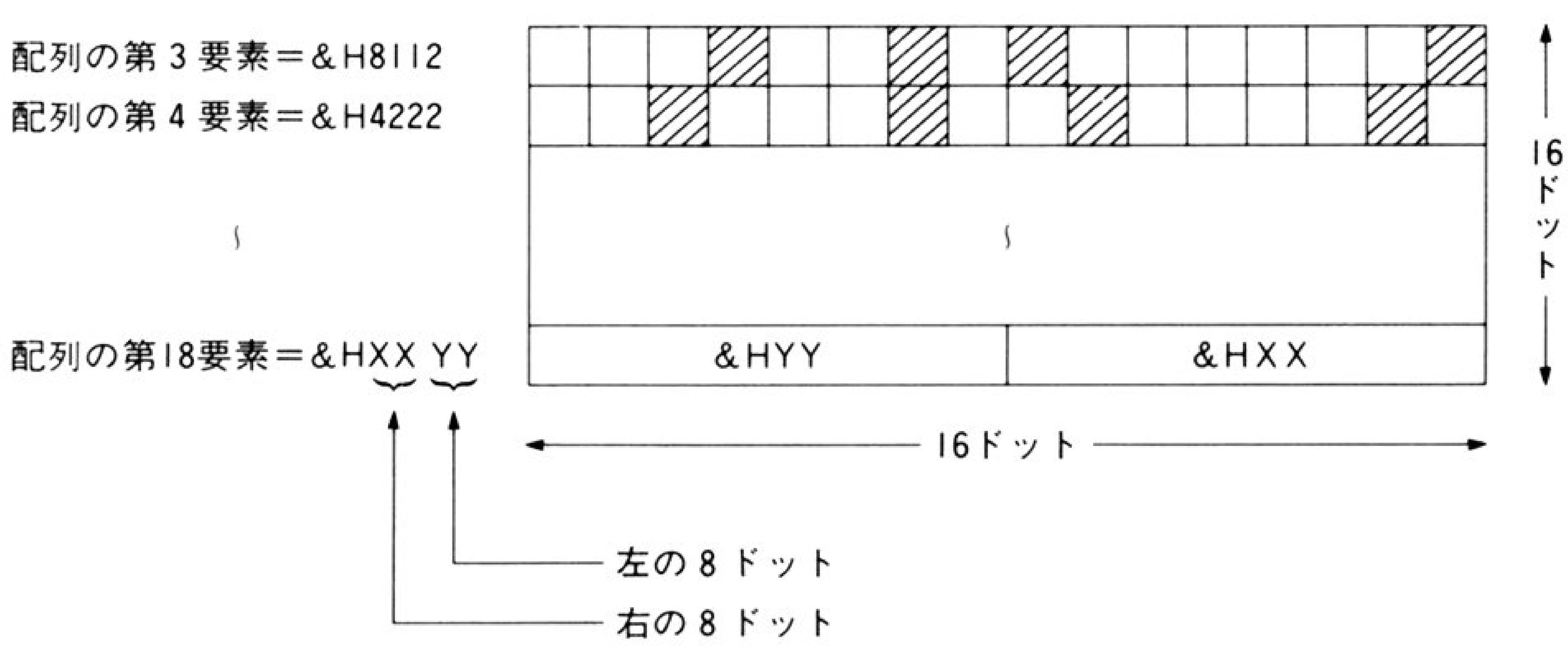
〈漢字コード〉としては，&H7621 から&H767E および&H7721 から&H777E までの値が指定可能です．〈漢字コード〉がこの範囲にないときは，〝Illegal function call〞エラーとなります．

〈整数型配列名〉の指定には，配列の要素番号を指定することはできません．また，配列内に格納されている文字パターンは以下のような形式でなければなりません．

配列の第 1 要素＝16

配列の第 2 要素＝16

配列の第 3 要素から第18要素までに，以下のように実際の文字パターンのドットイメージを格納してください．




注意：〈漢字コード〉の範囲は，機種により&H7621～&H765F の範囲に制約されるものがありますが，BASIC は&H7621～&H767E，&H7721～&H777E の範囲チェックしかしませんので注意が必要です．

**サンプルプログラム**

```
100 DIM CHRPTN%(17)
110 CHRPTN%(0)=16:CHRPTN%(1)=16
120 FOR I=2 TO 17:CHRPTN%(I)=&H1111:NEXT I
130 KPLOAD &H762A,CHRPTN%
140 PRINT KNJ$("1B4B")+KNJ$("762A")+KNJ$("1B48")
```

# LET

---

機　能　　変数に値を代入します.

書　式　　〔LET〕〈変数名〉=〈式〉

文　例　　LET I=I+1

I=(I+1)*J

A$=STRING$(L1, "*")

解　説　　キーワード LET はなくてもかまいません.

〈式〉の内容は, 数値, 文字列のいかなる型であってもかまいません. ただし, 文字型から数値型への代入またはその逆は許されません. 型の異なる数値型変数への代入では左辺の精度に合わせた代入が行われます.

L

# LINE

**機　能**　指定された2点間に直線を引きます.

**書　式**　LINE〔 | $(Wx_1,\ Wy_1)$ / STEP$(x_1,\ y_1)$ | 〕− | $(Wx_2,\ Wy_2)$ / STEP$(x_2,\ y_2)$ | 〔,〈パレット番号1〉〕,〔, | B / BF | 〕

〔, | 〈ラインスタイル〉 / 〈パレット番号2〉 / 〈タイルストリング〉 | 〕

**文　例**　LINE (100, 100)− (135, 100),7,, &HF99F

**解　説**

ワールド座標系の2点$(Wx_1,\ Wy_1)$と$(Wx_2,\ Wy_2)$を結ぶ直線を描きます. $(Wx_1,\ Wy_1)$が省略された場合, LPの値を採用します. STEPを付けると相対座標による指定になります.

引く線の色は〈パレット番号1〉により指定し, もしこれが省略された場合, 現在COLOR文によって設定されている, グラフィック画面のフォアグラウンドカラーが採用されることになります.

次のパラメータには, BかBFのどちらかを指定することができます. BはBoxの意味で$(Wx_1, Wy_1)$と$(Wx_2, Wy_2)$の2点を対角とする四角形の箱を描き, BFはBox Fillの意味でその四角形の内部を塗りつぶします. BF指定のときは, 〈ラインスタイル〉を指定することはできません. また, 〈パレット番号2〉, 〈タイルストリング〉はBF指定のときにだけ, 指定可能です.

〈ラインスタイル〉は, 引く線の形態を設定するオプションパラメータであり, 16ビットで表わすことのできる数, &H0から&HFFFF(16進)の値をとることができます. この16進の数値とラインの形態との関係は次の図に示す通りで, 16進の数の16ビットと画面上の16ビットが1対1に対応しており, 〝1〞の立っているビットに対応するドットが表示され, 〝0〞のビットに対応するドットは表示されません. この値は別に16進数でなくてもよいのですが, 16ビットに対応させて考える場合, 16進数の方がわかりやすく一般的です. また線の長さを16ビット以上引いた場合, このラインスタイルが画面上の16ビットごとに繰り返されることになります.

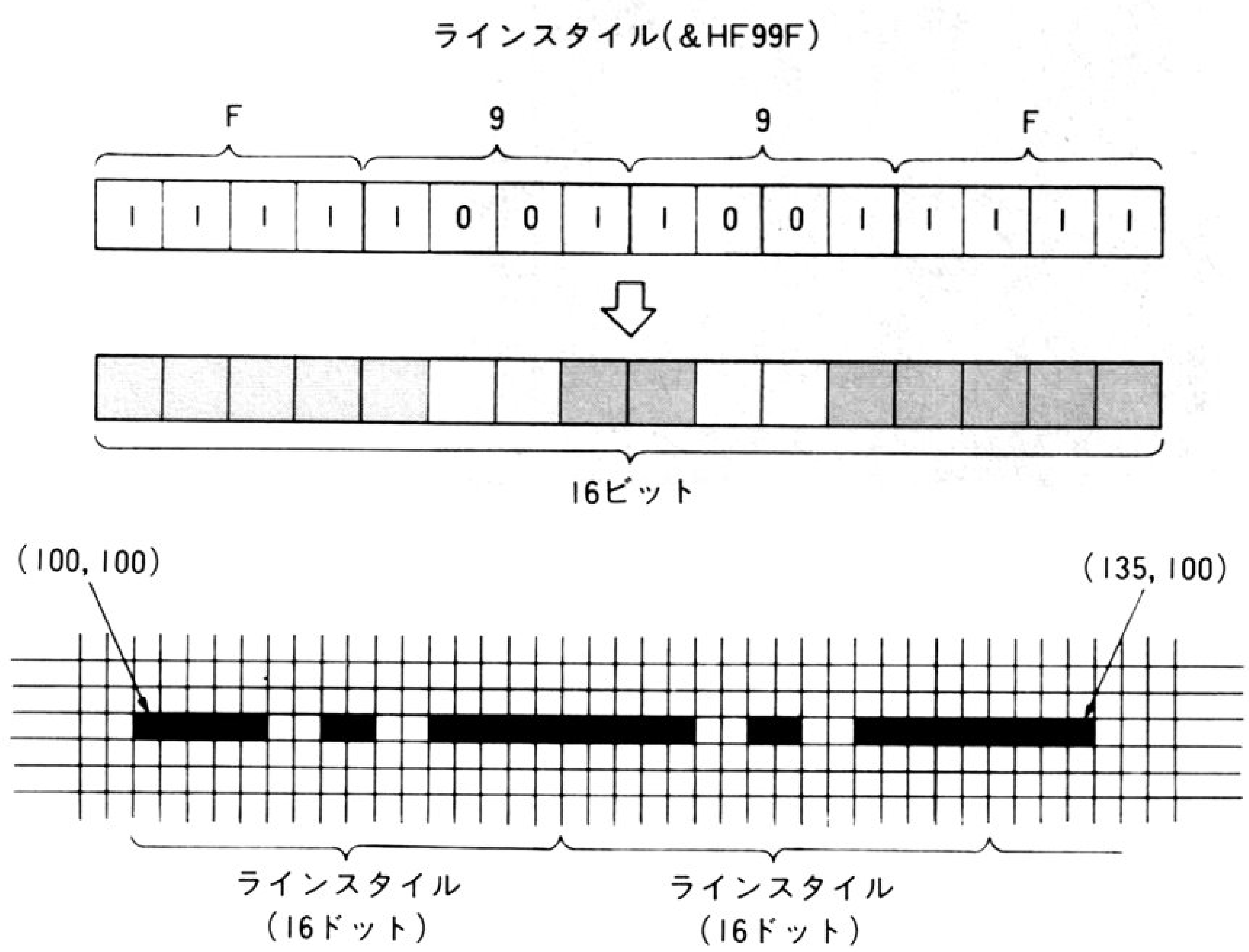
次の図は文例を実行した場合のものですが, 水平座標の100から135までの36ドットの間に, 1点鎖線のラインスタイルが繰り返されているのがわかります.

このようにラインスタイルを指定すると, 16ビット単位で線の形態を決めることができますから, 折れ線グラフや図面の作成の際, 点線や1点鎖線など簡単に

用いることができます．〈ラインスタイル〉が省略された場合は，ノーマルな直線を引きます．

〈パレット番号２〉は，四角形の内部を塗りつぶす際の色を，〈タイルストリング〉は，塗りつぶす際の模様を指定します．BF 指定があり，〈パレット番号２〉および〈タイルストリング〉共に省略された場合には，四角形を描いたのと同じ色で内部を塗りつぶします．

〈タイルストリング〉については，〔２〕PAINT を参照してください．



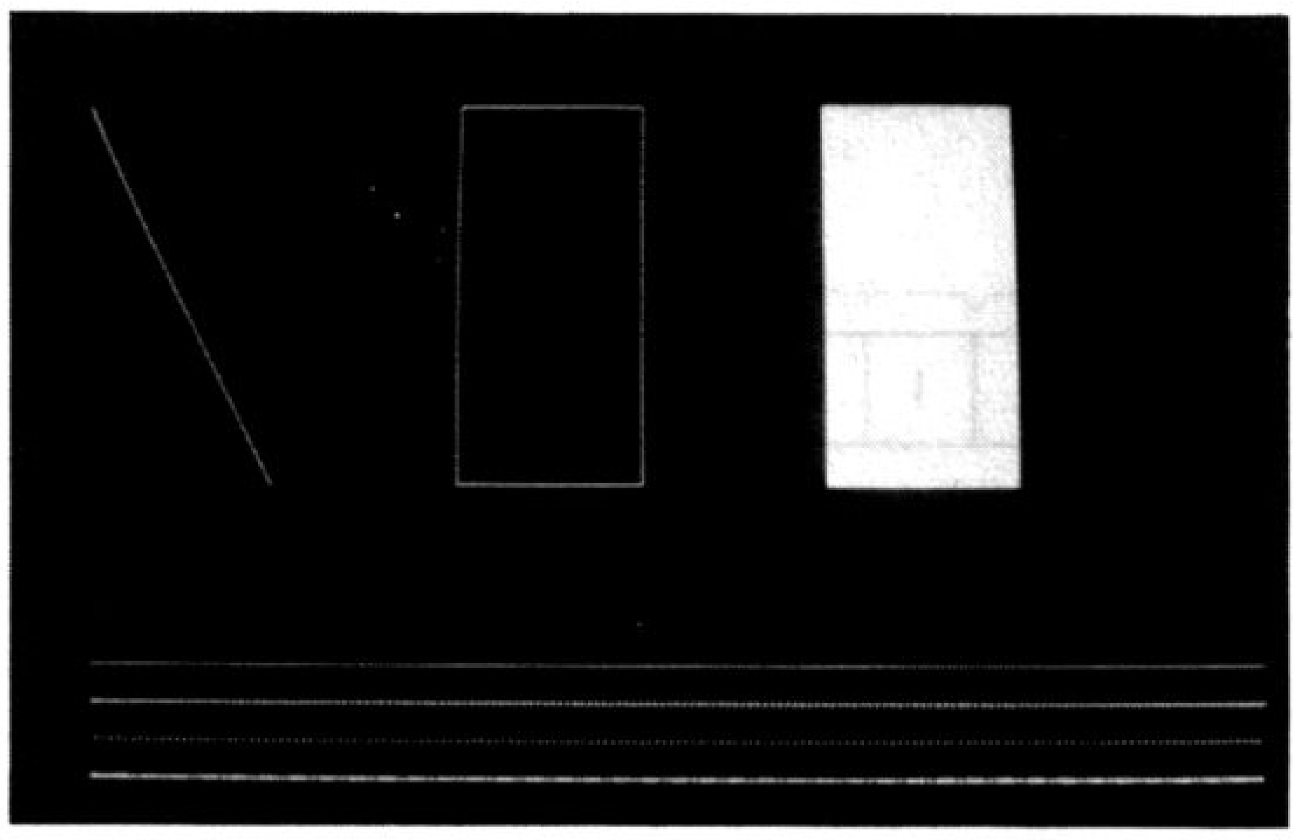
LINE(100, 100)−(135, 100), 7, , &HF99Fを実行した時

LINE 文を実行した場合，LP は（$Wx_2$，$Wy_2$）に移動します．

参照：〔２〕PAINT

サンプルプログラム

```
100 SCREEN 0,0,0,1
110 LINE(0,0)-(100,100),1
120 LINE(200,0)-(300,100),2,B
130 LINE(400,0)-(500,100),3,BF
140 '
150 LINE(0,150)-STEP(639,0),4,,&HAAAA
160 LINE(0,160)-STEP(639,0),5,,&HCCCC
170 LINE(0,170)-STEP(639,0),6,,&H8888
180 LINE(0,180)-STEP(639,0),7,,&HF99F
190 END
```



# LINE INPUT

**機能** キーボードから入力されるデータ(255文字以内)を，区切ることなく一括して文字変数に代入します．

**書式** LINE INPUT〔<プロンプト文>；〕<文字変数>

**文例** LINE INPUT "NAME"；NA$

**解説** <プロンプト文>は，入力を受ける前に画面に表示されるメッセージです．クエッションマーク(？)は表示されません．LINE INPUT 文においては入力されたすべてのデータがコンマなどの区切り記号も含めて<文字変数>に代入されます(INPUT 文の場合は区切り記号で区切られたデータの１つ１つが順次抽出され，変数に代入されます)．

LINE INPUT 文の実行は CTRL-C または STOP キーの入力によって中断することができます．その場合，STOP ON および STOP STOP の状態にない限りプログラムの実行も中断され，コマンドレベルにもどります．また，それまで入力した文字はすべて無効になります(文字変数には値は代入されません)．

参照：LINE INPUT #，INPUT

サンプルプログラム

```
100 INPUT "INPUT : ";A$
110 LINE INPUT "LINE INPUT : ? ";B$
120 PRINT:PRINT "INPUT : ",A$
130 PRINT "LINE INPUT : ",B$


run
INPUT : ? ASAD,ASDA,QE21,KJHH
?Redo from start
INPUT : ? ASDASD23QD
LINE INPUT : ? AS,DF,HJGJ,JHKJ,HJKJH

INPUT :        ASDASD23QD
LINE INPUT :   AS,DF,HJGJ,JHKJ,HJKJH
Ok
```

# LINE INPUT #

**機　能**　シーケンシャルディスクファイルより CR コードまでのデータを一括して文字変数に読み込みます．

**書　式**　LINE INPUT #〈ファイル番号〉，〈文字変数〉

**文　例**　LINE INPUT #1，A$

**解　説**

〈ファイル番号〉は，OPEN 文によってそのファイルを入力用としてオープンしたときに指定した番号です．〈文字変数〉は，行全体が代入される文字変数名です．LINE INPUT #文は，シーケルシャルファイルより，キャリッジリターンまでの，すべての文字を読みます．もし次に LINE INPUT #があればそのキャリッジリターンと続くラインフィードの組を飛ばして，次のキャリッジリターンに至るまでのすべての文字を読みます．LINE INPUT #文は，データファイルのそれぞれの行がいくつかの欄に分割されているときや，アスキーモードでセーブしたプログラムを他のプログラムでデータとして読み込むときなどに特に有効です．

参照：OPEN，INPUT #

# LINE INPUT WAIT

機 能　キーボードからデータを入力し，変数に代入します．その際，入力待ち時間を制限することができます．

書 式　LINE INPUT WAIT 〈待ち時間〉，〔〈プロンプト文〉 { ; , } 〕〈文字変数〉

文 例　LINE INPUT WAIT 50, "NO=", NO$

解 説　LINE INPUT WAIT は時間制限付きの LINE INPUT です．したがって LINE INPUT 文と INPUT WAIT 文の両方の機能を兼ねそなえています．詳しくはそれぞれの命令を参照してください．

参照：LINE INPUT, INPUT WAIT

サンプルプログラム

```
100 '-- タイプ゜の練習  --
110 *ENTRY
120 A$=""
130 FOR I=1 TO 3
140 A$=A$+CHR$(INT(RND*26+65))
150 NEXT I
160 PRINT "Problem : ";A$
170 LINE INPUT WAIT 60,B$:GOTO 200
180 PRINT:PRINT "Time out !"
190 GOTO *ENTRY
200 IF A$=B$ THEN PRINT "Right!":GOTO *ENTRY
210 PRINT "Wrong!"
220 GOTO *ENTRY

run
Problem : APZ
APZ
Right!
Problem : ZKW
ZK
Wrong!
Problem : WUX
WUX
Right!
Problem : JDD
^C
Break in 170
Ok
```

# LIST／LLIST

**機　能**　メモリ内にあるプログラムの全部または一部を表示／印刷します．

**書　式**　1)　LIST〔<行番号>〕〔－<行番号>〕

2)　LLIST〔<行番号>〕〔－<行番号>〕

**文　例**　LIST 100-200

LIST .

**解　説**　プログラムの内容を，書式1) は画面に，書式2) はプリンタに出力します．LIST コマンドは実行し終ると，いつでもコマンドレベルに戻ります．

<行番号>が省略された場合は，最も若い行番号のプログラム行よりリストが始まります(リストはプログラムの終り，またはストップキーの入力により終ります)．最初の行番号のみが指定された場合には，その指定された行だけがリストされます．最初の行番号とそれに続くマイナス記号までが指定された場合は，その行番号に始まり，それよりも大きい行番号の行すべてがリストされます．マイナス記号とそれに続く 2 番目の行番号だけが指定された場合には，プログラムの始めからその行番号までの行がリストされます．両方の行番号がマイナス記号でつながれて指定された場合には，その範囲のすべての行がリストされます．

<行番号>に〝.〃(ピリオド)を指定すると BASIC インタプリタ内のポインタが示している現在の行を指します(たとえば，最後に修正された行)．

# LOAD

| | |
|---|---|
| 機　能 | **プログラムをメモリにロードします.** |
| 書　式 | LOAD <ファイルディスクリプタ>〔, R〕 |
| 文　例 | LOAD "2 : DEMO" |
| | LOAD "COM : N82NN" |

解　説

<ファイルディスクリプタ>で指定されたプログラムファイルをメモリ上にロードします. ロードを実行すると, すべての開いているファイルが閉じられ, 変数はその値が失われますが, Rオプションをつけると, ファイルは開いたままで, プログラムをロード後, ただちに実行を開始します.

LOAD文が使用できるファイル機器はディスクファイル, RS-232Cコミュニケーションファイル, キーボードファイルの3種です.

**注意**: RS-232Cコミュニケーションポートからのプログラムロードの場合, プログラムのロードが完了しても, 自動的にLOADコマンドは終了しません. 強制BREAK(STOPキーによる)が必要です.

# LOAD ?

|機　能| カセットに正しくプログラムがセーブできたかを確認します.

|書　式| LOAD ? <ファイルディスクリプタ>

|文　例| LOAD ? "CAS2 : TEST"

|解　説| 〈ファイルディスクリプタ〉により指定されたプログラムファイルの内容と，メモリ中のプログラムの内容が同一であるかを調べます．両者が異なっている場合には "Bad" と表示されます．

このコマンドは，通常SAVEの直後に，正しくセーブが行われたことを確認する目的で用います．またこのコマンドは，カセットファイルに対してのみ有効であり，他の装置上のファイルが指定された場合には通常のLOADコマンドと全く同じ動作をします．カセットファイルが指定された場合には比較を行うのみで実際のロード動作は行いませんから，メモリ中のプログラムは失なわれません．

# LOCATE

| | |
|---|---|
| 機能 | **テキスト画面のカーソルの位置を制御します.** |
| 書式 | LOCATE〔<X>〕〔, <Y>〕〔, <カーソルスイッチ>〕 |
| 文例 | LOCATE 10, 10<br>LOCATE ,,0 |

解説

カーソルをテキスト画面のキャラクタ座標(X, Y)の位置へ移動します.

<X>は水平座標を表わし，省略した場合は0とみなされます．<Y>は垂直座標を表わし，省略された場合は現在カーソルがセットされている行とみなされます. これらは共に画面の左上を(0, 0)とするキャラクタ座標により指定します.

<カーソルスイッチ>はカーソルの表示状態を決めるスイッチで，0だと表示されなくなり，1で表示されます.

**注意**：水平座標が奇数桁の場合，日本語の表示あるいは日本語入力を正しく行うことはできません.

# LSET／RSET

**機　能**　ファイルバッファにデータを設定します．

**書　式**　LSET <文字型変数>＝<文字列>

RSET <文字型変数>＝<文字列>

**文　例**　LSET A$＝〝単価〃

RSET A$＝MKS$(I)

**解　説**　Left SET，Right SET の意味で FIELD 文により，ランダムアクセスバッファ上に割当てられた文字型変数に対し，指定された文字列を左づめ，あるいは右づめで設定します．格納するデータは文字型データでなければならず，数値データは MKI$，MKS$，MKD$のいずれかにより文字形データに変換しておかなければなりません．

<文字列> の文字数が FIELD 文で割り当てられた長さより短い場合，LSET では左詰め，RSET では右詰めでフィールドを満たします．また，<文字列>がフィールドより長い場合は，LSET，RSET とも右側の部分が失われます．

<文字型変数>は，FIELD 文であらかじめ定義されていなければなりません．定義されていない変数を用いるとエラーが発生します．

参照：FIELD，PUT，GET

# MERGE　　（DISK モード）

機　能　メモリ上のプログラムとファイルから入力したプログラムを混合します．

書　式　MERGE 〈ファイルディスクリプタ〉

文　例　MERGE "2 : TEST"

解　説

メモリ上のプログラムと〈ファイルディスクリプタ〉により指定したプログラムファイルを1つにして，メモリ上に置きます．ここで，指定されたファイルはアスキーセーブされていなければなりません．そうでない場合には，エラーになります．ファイル中のプログラムと，メモリ中のプログラムに同一の行番号があった場合には，ファイル中の行でメモリ中の行を置き換えます．

MERGE コマンドは実行を終了すると，常にコマンドレベルに戻ります．

参照：SAVE

# MID$

| | |
|---|---|
| 機　能 | **文字列の一部を置き換えます.** |
| 書　式 | MID$ (〈**文字列1**〉, 〈**式1**〉〔, 〈**式2**〉〕) = 〈**文字列2**〉 |
| 文　例 | MID$ (A$, 3) = "ABC" |

解　説

〈文字列1〉の〈式1〉番目の文字から〈式2〉個の文字を, 〈文字列2〉の最初から〈式2〉個の文字列で置き換えます.

〈式2〉が省略された場合, または〈式2〉の値が〈文字列2〉の文字数を越える場合は, 〈文字列2〉のすべての文字と置き換わります.

〈式1〉+〈式2〉−1の値が〈文字列1〉の文字数より大きくなり, 指定した文字列があふれてしまう場合は, (〈文字列1〉の文字数) −〈式1〉+1の値を〈式2〉とし, 〈文字列2〉の残りの部分は無視します.

また当然のことながら, 〈式1〉の値は〈文字列1〉の文字数より大きくてはいけないし, 0以下であってもいけません. したがって〈文字列1〉にヌルストリングを指定することはできません.

参照：MID$関数

サンプルプログラム

```
100 DATA "My name is !!!!!!!!
110 DATA Sato,Tanaka,Suzuki,Murakami
120 READ A$
130 FOR I=1 TO 4
140  READ NAM$:MID$(A$,12)=NAM$
150  PRINT A$
160 NEXT I


run
My name is Sato!!!!
My name is Tanaka!!
My name is Suzuki!!
My name is Murakami
Ok
```

# MON

**機能** モニタモードに入ります.

**書式** MON

**文例** MON

**解説** BASIC モードからモニタモードに制御を移行します. モニタモードについての詳細はユーザーズマニュアルを参照してください. CTRL-B で, もとの BASIC モードにもどります.

# MOTOR

**機 能**　カセットテープレコーダのモータのON，OFFを制御します．

**書 式**　MOTOR〔〈スイッチ〉〕

**文 例**　MOTOR 1

**解 説**　〈スイッチ〉を0にすればモータはOFFとなり，0以外の値にすればONとなります．また，〈スイッチ〉を指定しない場合，モータがONの状態ならばOFFに，OFFの状態ならばONにします．

**サンプルプログラム**

```
100 MOTOR 1
110 PRINT "Rewind and set the tape."
120 INPUT "Are you ready (y/n):",A$
130 IF LEFT$(A$,1)<>"y" THEN END
140 PRINT
150 MOTOR 0
160 PRINT "Push PLAY button."
170 INPUT "Are you ready (y/n):",A$
180 IF LEFT$(A$,1)<>"y" THEN END
190 MOTOR
200 END

run
Rewind and set the tape.
Are you ready (y/n):y

Push PLAY button.
Are you ready (y/n):y
Ok
```

# NAME　　(DISK モード)

**機　能**　**ディスクファイルの名前を変更します.**

**書　式**　NAME 〈旧ファイル名〉 AS 〈新ファイル名〉

**文　例**　NAME "SAMPLE" AS "SAMPLE 1"

NAME "2 : DATA.BAS" AS "2 : DATAI"

**解　説**　〈旧ファイル名〉で表わされているディスク上のファイルを〈新ファイル名〉に変更します.

ファイルの指定形式は次の通りです.

**〔ドライブ番号 : 〕 ファイル名**

ドライブ番号が省略された場合, ドライブ1のディスクが処理の対象になります.

〈旧ファイル名〉と〈新ファイル名〉に指定されるドライブ番号は同じでなければなりません.

なおファイル名の変更を行うファイルはクローズされた状態でなければなりません.

# NEW

**機　能**　メモリにあるプログラムを抹消し，すべての変数をクリアします．

**書　式**　NEW

**文　例**　NEW

**解　説**　NEW コマンドはプログラムを抹消し，すべての変数をクリアします．NEW コマンドの実行が終わると，いつもコマンドレベルに戻ります．

NEW コマンドは，開かれているファイルがある場合は，それを自動的にすべて閉じます．

# NEW ON

**機　能**　システムを再起動します．

**書　式**　NEW ON 〈式〉

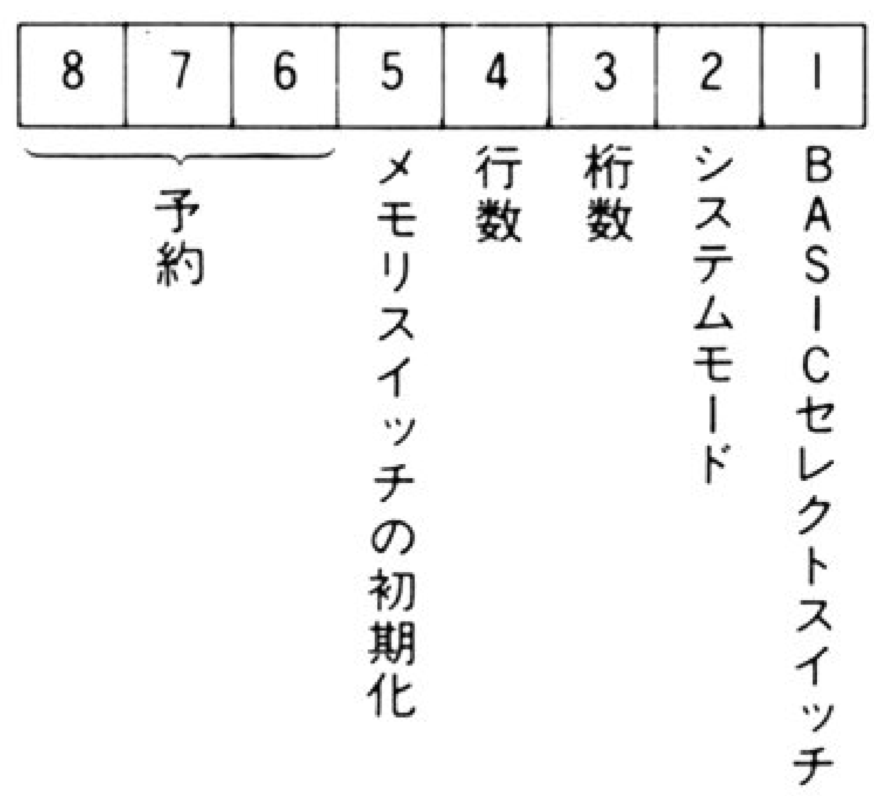
**文　例**　NEW ON 1

**解　説**

NEW ON は指定された値によりシステムを再起動します．

〈式〉の値は，ディップスイッチ SW2 の各ビットに対応して意味付けられており，8 ビットで値を指定します（ディップスイッチ SW2 の詳細に関しては，ユーザーズマニュアルを参照してください）．

ディップスイッチSW2の機能




| | | |
|---|---|---|
| 〈BASICセレクトスイッチ〉 | 0 | ：N88-BASIC |
| | 1 | ：N-BASIC |
| 〈システムモード〉 | 0 | ：BASICモード |
| | 1 | ：ターミナルモード |
| 〈桁数〉 | 1 | ：1行80桁 |
| | 0 | ：1行40桁 |
| 〈行数〉 | 1 | ：1画面25行 |
| | 0 | ：1画面20行 |
| 〈メモリスイッチの初期化〉 | 0 | ：システム既定値で初期化． |
| | 1 | ：メモリスイッチを有効にする． |

スイッチ番号 1 はディップスイッチでは意味がありませんが，NEW ON での指定は BASIC の選択を指定します．N-BASIC を指定した場合は，ディスク装置に N-BASIC(86)のシステムディスクを装着しておく必要があります．

**注意**：スイッチ番号 5 ～ 8 は NEW ON コマンドでは意味を持ちません．

サンプルプログラム

`new on 0` $N_{88}$-BASICが再起動されます．その際のモードは次のとおりです．

- BASICモード
- 1行40桁
- 1画面20行

`new on 1` N-BASICが起動されます．その際のモードは次のとおりです．

- 1行40桁
- 1画面20行

`new on 2` $N_{88}$-BASICがターミナルモードで起動されます．その際のモードは次のとおりです．

- 1行40桁
- 1画面20行

# ON COM GOSUB



機　能　通信回線からの割り込みが発生した時，分岐する処理ルーチンの開始行を定義します．

書　式　ON COM〔(<回線番号>)〕GOSUB <行番号>

文　例　ON COM GOSUB 1000

ON COM(2) GOSUB *RECV

解　説　指定した通信回線から RS-232C のコミュニケーションポートに入力割り込みがあった時に分岐する処理ルーチンの開始行番号を定義します．

<回線番号> に指定できる値と意味は次のとおりです．

1：RS-232C 第 1 回線
2：RS-232C 第 2 回線 ｝専用の拡張インタフェースボードが必要です．
3：RS-232C 第 3 回線 ｝

<回線番号>は省略することができます．省略された場合，第 1 回線が指定されたものとみなします．

<行番号> とは分岐させる処理ルーチンの開始行番号です．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで RETURN 命令によって行います．単に RETURN とした時は，中断した所から再開し，後ろに行番号を指定した時は，その行から再開します．

**注意**：この命令によって割り込み処理ルーチンに分岐させるには，割り込みが発生した時，COM ON の状態でなければなりません．

ON COM GOSUB は，RS-232C コミュニケーションポートを OPEN した後実行されなければ有効となりません．

**参照**：COM ON／OFF／STOP，OPEN

サンプルプログラム

```
100 OPEN "COM:N82NN" AS 1
110 ON COM GOSUB *RECIEVE
120 COM ON
130 PRINT TIME$:GOTO 130
140 *RECIEVE
150 IF LOC(1)<>0 THEN PRINT INPUT$(LOC(1),#1);
160 RETURN
```

# ON ERROR GOTO

| | |
|---|---|
| 機　能 | **エラートラップを可能にし，エラー処理ルーチンの開始行を定義します．** |
| 書　式 | ON ERROR GOTO <行番号> |
| 文　例 | ON ERROR GOTO 1000 |

解　説　エラートラップ機能が可能になると，エラーが検出された時指定されたエラー処理ルーチンへプログラムの制御が移ります．もし<行番号>の行が存在しなければ，〝Undefined line number〞エラーが起こります．エラートラップ機能を禁止するには，ON ERROR GOTO 0 を実行してください．その後のエラーはエラーメッセージを表示し，プログラムの実行は停止します．また，エラー処理ルーチンに ON ERROR GOTO 0 の文がある場合には，BASIC はエラートラップの原因となったエラーのエラーメッセージを表示し，プログラムの実行を停止します．

エラー処理ルーチン中でエラー回復処理を行わないエラーをトラップした場合には，ON ERROR GOTO 0 を実行することをお勧めします．

**注意**：エラー処理サブルーチンが実行中にエラーが起きた場合には，それに対するエラーメッセージが表示され，実行は停止します．エラー処理サブルーチンの中ではエラートラップは起こりません．

**参照**：RESUME

サンプルプログラム

```
100 '  XのY乗を求める.
110 ON ERROR GOTO *ERRORMES
120 *START:INPUT "X : ";X:INPUT "Y : ";Y
130   IF X=0 THEN ERROR 250
140  Z=X^Y:PRINT "  Y":PRINT"  X =";Z
150 GOTO 120
160 *ERRORMES
170  IF ERR=250 THEN PRINT "定義されません."
180  IF ERR=6 THEN PRINT "オーハ゛ーフローです."
190 RESUME *START
```

```
run
X : ? 2
Y : ? 10
  Y
  X = 1024
X : ? 100
Y : ? 100
オーハ゛ーフローです.
X : ? 0
Y : ? -2
定義されません.
X : ?
Break in 120
Ok
```

# ON HELP GOSUB

機能　ヘルプキーによる割り込みルーチンの開始行を定義します．

書式　ON HELP GOSUB <行番号>

文例　ON HELP GOSUB 1000

解説　ヘルプキーを押した時に分岐する処理ルーチンの開始行を定義します．

ON HELP GOSUBはファンクションキーによる割り込みの1つであり，特殊な用途のためのON KEY GOSUBだと考えることができます．したがって，その使い方も同じです．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じでRETURN命令によって行います．単にRETURNとした時は中断した所から再開し，後ろに<行番号>を指定した時は，その行から再開します．INPUT文実行中に割り込んだ場合は，単にRETURNで復帰させるとINPUT文の次の文から実行を再開しますから，行番号を指定するか，プログラムで適切な処置をしなければなりません．

本来この命令は，給与計算などのアプリケーションプログラムを作成した際，プログラマーが，そのプログラムの使い方，入力の仕方などを説明したインストラクションルーチンを定義しておき，ユーザーがそのプログラム実行中に，次に何をしてよいのかわからなくなった時にヘルプキーによる割り込みで指示を受ける，といったような用途のために用意されているものです．

**注意**：この命令によって割り込み処理ルーチンに分岐させるには，割り込みがあった時HELP ONの状態でなければなりません．

**参照**：HELP ON／OFF／STOP，ON KEY GOSUB

サンプルプログラム

```
100 '－－　ＨＥＬＰキーを使ったメッセージ゛フ゜リント　－－
110  ON HELP GOSUB *MESSAGE
120  HELP ON
130 '
140  FOR I=1 TO 10
150   INPUT N
160   PRINT "N =";N,"N*N =";N*N
170  NEXT I
180  HELP OFF:END
190 '-- instruction --
200 *MESSAGE
210  HELP OFF
220  PRINT
230  PRINT "数字を入力してください．"
240  PRINT "その２乗を計算します．"
250  HELP ON
260 RETURN 140
```

```
run
? ABC
?Redo from start
?

数字を入力してください．
その２乗を計算します．
? 10
N = 10          N*N = 100
? 12.45
N = 12.45       N*N = 155.003
? 10,23
?Redo from start
?

数字を入力してください．
その２乗を計算します．
?
Break in 150
Ok
```

# ON…GOSUB／ON…GOTO

| | |
|---|---|
| 機　能 | **指定されたいずれかの行に分岐します.** |
| 書　式 | ON <式> GOSUB <行番号>〔, <行番号> …〕<br>ON <式> GOTO <行番号>〔, <行番号> …〕 |
| 文　例 | ON A GOSUB 100, 200, 300, 400<br>ON A GOTO 100, 200, 300, 400 |

解　説

<式>の値がプログラムのどの行番号の行に分岐するかを決定します. <行番号>の並びは1から始まる数に対応します. たとえば, <式>の値が3であったなら, 行番号の並びの3番目の行が分岐先となります.

ON…GOSUB文では, 並びの各行番号はサブルーチンの最初の行の行番号でなければなりません.

<式>の値が負になった場合には, "Illegal function call"エラーが起こりますが, 値が0または並びの行番号の個数より大きくなった場合には, 次の行へプログラムの制御が移り, エラーは起こりません.

サンプルプログラム

```
100 *ENTRY
110  INPUT "N=";N
120  SG=SGN(N)+1
130  ON SG GOTO *ZERO,*PLUS
140  PRINT "MINUS!":GOTO *ENTRY
150 *ZERO:PRINT "ZERO!":GOTO *ENTRY
160 *PLUS:PRINT "PLUS!"
170  GOTO *ENTRY

run
N=? 12
PLUS!
N=? 45
PLUS!
N=? 0
ZERO!
N=? -32
MINUS!
N=? -1.25
MINUS!
N=?
Break in 110
Ok
```

# ON KEY GOSUB

| | |
|---|---|
| 機　能 | **ファンクションキーによる割り込みルーチンの開始行を定義します.** |
| 書　式 | ON KEY GOSUB 〈行番号〉〔, 〈行番号〉 …〕 |
| 文　例 | ON KEY GOSUB 100, 200, 300, 400 |

解　説　ファンクションキーを押した時に分岐する処理ルーチンの開始行を定義します.

この命令は，割り込みが発生した時(キーが押された時)にどの処理ルーチンに飛ぶかを定義するだけのものであり，この命令が実行されても割り込みが起こることはありません.割り込みが発生するのは，あくまで何らかの外的要因(この場合，キーが押されたこと)があった時です.

〈行番号〉とは，割り込みが発生した時に分岐させる処理ルーチンの開始行番号で，この並びの順序は，ファンクションキーのキー番号と1対1に対応していなければなりません(たとえば,ファンクションキー1とファンクションキー3をキー割り込みの対象としたい場合, ON KEY GOSUB 100,,300 と指定します). PC-9800 シリーズパーソナルコンピュータがサポートしているファンクションキーは全部で10番までありますから，最大10個の〈行番号〉を並べて書くことができます.もちろん実際の行番号の代わりにラベル名を使うこともできます.

KEY ON 状態になっているファンクションキーが押されても,対応する割り込みルーチンがON KEY GOSUB 命令で定義されていない場合，割り込みは発生しません.たとえば,

```
10 ON KEY GOSUB 100,,200
20 KEY (2) ON
```

の場合, ファンクションキー2は KEY ON 状態になっていますが, 割り込みルーチンの定義がなされていないので割り込みは発生しません.

処理ルーチンからの復帰は，一般にサブルーチンの場合と同じで RETURN 命令によって行います.単に RETURN とした時は，中断した所から再開し，後ろに行番号を指定した時は，その行から再開します.

**注意**：この命令によって割り込み処理ルーチンに分岐させるには，割り込みが発生した時，KEY ON の状態でなければなりません.

**参照**：KEY ON／OFF／STOP

サンプルプログラム

```
100 '-- sin , cos --
110  ON KEY GOSUB *SINSUB,*COSSUB,*ENDSUB
120  FOR I=1 TO 4
130   KEY(I)ON
140  NEXT I
150  DEG=0:PI=3.14159
160 *START
170   TH=DEG/180*PI
180   PRINT "TH =";DEG;
190   IF FLAG=1 THEN *COSC
200   PRINT TAB(12);"SIN(TH) =";SIN(TH):GOTO *EXIT1
210 *COSC:PRINT TAB(35);"COS(TH)=";COS(TH)
220 *EXIT1:DEG=DEG+10
230  GOTO *START
240 '-- key interrupt routine --
250 *SINSUB:FLAG=0:RETURN
260 *COSSUB:FLAG=1:RETURN
270 *ENDSUB:KEY OFF:END
```

```
run
TH = 0       SIN(TH) = 0
TH = 10      SIN(TH) = .173648
TH = 20      SIN(TH) = .34202
TH = 30      SIN(TH) = .5
TH = 40      SIN(TH) = .642787
TH = 50      SIN(TH) = .766044
TH = 60      SIN(TH) = .866025
TH = 70      SIN(TH) = .939692
TH = 80                               COS(TH)= .173649
TH = 90                               COS(TH)= 1.12352E-06
TH = 100                              COS(TH)=-.173647
TH = 110                              COS(TH)=-.342019
TH = 120     SIN(TH) = .866026
TH = 130     SIN(TH) = .766046
TH = 140     SIN(TH) = .642789
TH = 150     SIN(TH) = .500002
TH = 160     SIN(TH) = .342022
TH = 170     SIN(TH) = .173651
Ok
```

# ON PEN GOSUB

機能　ライトペンによる割り込み処理ルーチンの開始行を定義します.

書式　ON PEN GOSUB <行番号>

文例　ON PEN GOSUB ＊PENTEST

解説　ライトペンが押されて光を感じた時，分岐する処理ルーチンの開始行を定義します.

この命令によって処理ルーチンに分岐させるには，PEN ON の状態でなければなりません.

この命令は対象がライトペンであることを除けば，他の割り込み定義命令（KEY，COM，HELP，STOP）と同様です.

参照：ON KEY GOSUB etc.，PEN ON／OFF／STOP

サンプルプログラム

```
100 COLOR 0,7:CLS 2
110 ON PEN GOSUB *PENSUB
120 PEN ON
130 GOTO 130
140 '-- PEN ROUTINE --
150 *PENSUB : PEN OFF
160 PRINT "trigger =";PEN(0),"x =";PEN(1),"y =";PEN(2)
170 FOR I=1 TO 3000:NEXT
180 PEN ON :RETURN
```

```
run
trigger = 0    x = 39         y = 18
trigger = 0    x = 29         y = 12
trigger = 0    x = 23         y = 14
trigger = 0    x = 28         y = 6
trigger = 0    x = 59         y = 5
trigger = 0    x = 59         y = 5
trigger = 0    x = 58         y = 5
^C
Break in 130
Ok
```

# ON STOP GOSUB

機能　STOP キーによる割り込み処理ルーチンの開始行を定義します.

書式　ON STOP GOSUB <行番号>

文例　ON STOP GOSUB 100

解説　STOP キーを押した時に, 割り込みによって分岐するルーチンの開始行を定義します.

この命令は割り込みの対象としてSTOP キーを使っていることを除けば, ファンクションキーによる割り込みと同じですから, ON KEY GOSUB の項を参照してください.

この命令はアプリケーションプログラムの実行中に, ユーザーが誤ってSTOP キーを押してしまい, プログラムの実行が中断されてしまうのを防止するためのものです. したがって不用意にこの命令を実行してしまうと本来のSTOP キー及びCTRL-C の機能は全く失われ, プログラムを中断することができなくなってしまいます. 動作ミスのないプログラムにしてから使用してください.

参照 : ON KEY GOSUB, STOP ON／OFF／STOP

サンプルプログラム

```
100 ON STOP GOSUB *MASK
110 STOP ON
120 F$="###     ##.######^^^^  ##.######^^^^"
130 PRINT "Degree";TAB(12);"Sin";TAB(27);"Cos"
140 FOR I=0 TO 180 STEP 20
150  TH=I/180*3.14159
160  PRINT USING F$;I;SIN(TH);COS(TH)
170 NEXT I:STOP OFF:END
180 *MASK:RETURN
```

```
run
Degree        Sin            Cos
  0      0.000000E+00   1.000000E+00
 20      3.420200E-01   9.396930E-01
 40      6.427870E-01   7.660450E-01
 60      8.660250E-01   5.000010E-01
 80      9.848070E-01   1.736490E-01
100      9.848080E-01  -1.736470E-01
120      8.660260E-01  -4.999990E-01
140      6.427890E-01  -7.660430E-01
160      3.420220E-01  -9.396920E-01
180      2.621550E-06  -1.000000E+00
Ok
```

# ON TIME$ GOSUB

機　能　**リアルタイムタイマの割り込み発生時刻と，その時分岐する処理ルーチンの開始行を定義します．**

書　式　ON TIME$=〝hh : mm : ss〃 GOSUB 〈行番号〉

文　例　ON TIME$=〝07 : 30 : 00〃 GOSUB ＊MORNINGCALL

解　説　割り込み時刻の設定の仕方はTIME$の場合と同様です．ただし，ここで設定する時刻はリアルタイムタイマに影響を与えません．またこの命令で制御できる分解能は2秒ですから，場合によっては±1秒程度の誤差がでることがあります．

〈行番号〉は割り込みによって分岐する処理ルーチンの開始行です．もちろんラベル名を使うこともできます．

ON TIME$ GOSUBは実行された時点で，設定された時刻から現在の時刻を引き算して割り込み時刻を決定します．したがって，もし現在の時刻をTIME$文で変更してから割り込み時刻を設定する場合は，TIME$ 文は必ずON TIME$ GOSUB文の直前で実行しなければなりません．

またこの命令によって割り込み処理ルーチンに分岐させるには，TIME$ ONの状態でなければなりません．

参照：TIME$ ON／OFF／STOP，ON KEY GOSUB，TIME$

サンプルプログラム

```
100 ON TIME$="07:30:00" GOSUB *MORNING
110 TIME$ ON
120 LOCATE 30,10:PRINT TIME$:GOTO 120
130 '-- Morning call routine --
140 *MORNING
150 FOR I=1 TO 10
160  BEEP:PRINT "Good Morning !!"
170  FOR J=0 TO 300:NEXT J
180 NEXT I:END
```

# OPEN

**機　能**　ファイルを開きます.

**書　式**　OPEN 〈ファイルディスクリプタ〉〔FOR 〈モード〉〕AS〔#〕〈ファイル番号〉

**文　例**　OPEN "2 : DATA1" AS # 1

OPEN "COM1 : N82NN" AS # 1

**解　説**　〈ファイルディスクリプタ〉で指定されたファイルを指定された〈ファイル番号〉でオープンします. 以後, オープンされたファイルへの入出力は, 〈ファイル番号〉を指示することにより行います.

〈モード〉はファイルへのアクセス方法を指定します. モードには次の 4 種類があります.

| | |
|---|---|
| INPUT | 既存のファイルから入力を行うことを指示します. |
| OUTPUT | 新しくファイルを作り, 出力を行うことを指示します. |
| APPEND | 既存のファイルの終りから追加を行うことを指示します. |
| 省略した時 | FOR 節が省略されると, ディスクファイルの場合ランダムアクセスを行うことを指示します. また, RS-232C コミュニケーションファイルの場合, 一般に省略します. |

INPUT, APPEND モードでは, 指定されたファイルが存在しないと, "File not found"のエラーとなります. OUTPUT モードでは, 常に指定された名前のファイルを新しく作り, 同一名のファイルがあった場合にはそのファイルは削除されます. ランダムアクセスでファイルが存在しない場合には新たに作られます.

〈ファイル番号〉は 1 から13までの値を用いることができますが, BASIC を使い始める時に宣言したファイルの数を超えてはなりません. また, 既にオープンされているファイルが利用している番号を用いることはできません.

OPEN は以後の入出力の際に用いるバッファ領域を確保し, ファイルが閉じられるまでの間, そのバッファは指定したファイルへの入出力操作専用に用いられます (FIELD 文参照).

RS-232C コミュニケーションファイルをオープンする際には, ファイル名 COM 〈回線番号〉につづけて, パリティ, ビット数等を指定する必要があります.

指定形式は次のとおりです.

"COM〔〈回線番号〉〕:〔〈パリティ〉〔〈ビット〉〔〈ストップビット〉
〔〈XONスイッチ〉〔〈Sパラメータ〉〕〕〕〕〕"

〈パリティ〉はE, O, Nで表わし, それぞれ偶数パリティ, 奇数パリティ, パリティ無しを意味します.

〈ビット〉は1文字を表わすのに用いるビット数で, 7または8で指定します. 7を指定した場合には, 必ず〈パリティ〉でOまたはEを指定しなければなりません. また8を指定した場合〈パリティ〉はNでなければなりません.

〈ストップビット〉はストップビット数を決定し, 1, 2, 3で指定します. 1は1ビット, 2は1.5ビット, 3は2ビットを意味します.

〈XONスイッチ〉はXON/XOFFによる通信制御を行うことを指定し, スイッチとしてXが指定されるとXON/XOFF制御を行います. Nが指定された場合にはこの制御は行われません.

〈Sパラメータ〉は〈ビット〉に7を指定した時に, キャラクタコード128以上の文字(カナ文字など)の入出力ができなくなるのを補助するパラメータです. 指定はSまたはNで行い, Sを指定した時入出力可能で, Nを指定した時は入出力不可能になります.

OPEN文で扱うことのできるファイル機器は次のとおりです.

| ファイル | 使用できるモード |
|---|---|
| ディスクファイル(ドライブ番号:) | INPUT, OUTPUT, APPEND, 省略 |
| RS-232Cコミュニケーションファイル(COM:) | 省　略 |
| キーボードファイル(KYBD:) | INPUT |
| スクリーンファイル(SCRN:) | OUTPUT |
| プリンタファイル(LPT:) | OUTPUT |
| カセットファイル(CAS:) | INPUT, OUTPUT |

# OPTION BASE

| | |
|---|---|
| 機能 | **配列の添字の下限を宣言します.** |
| 書式 | OPTION BASE \| 0 \| / \| 1 \| |
| 文例 | OPTION BASE 1 |

**解説**

配列の添字の下限を0または1にすることを宣言します．通常，この宣言が用いられない場合には，下限は0ですが，OPTION BASE 1を実行すると下限は1になり，以後，配列の添字として0が用いられると，〝Subscript out of range〟エラーが起こるようになります．

この宣言は，プログラム中で配列変数が宣言，または引用された後に行うことはできません．また一度宣言すると，再宣言によって下限を変更することはできません．このような場合には〝Duplicate Definition〟エラーになります．

OPTION BASEは一度宣言すると，RUNあるいはCLEARを行うまでは，解除，変更することができません．

サンプルプログラム

```
100 OPTION BASE 0
110 DIM A(1)
120 PRINT "A(0) =";A(0),"A(1) =";A(1)
130 '
140 CLEAR:OPTION BASE 1
150 '-- error occur --
160 PRINT "A(0) =";A(0),"A(1) =";A(1)


run
A(0) = 0       A(1) = 0
A(0) =
Subscript out of range in 160
Ok
```

# OUT

**機能**　コンピュータの出力ポートに1バイトのデータを送出します.

**書式**　OUT 〈ポート番号〉, 〈式〉

**文例**　OUT 64, 32

**解説**　〈ポート番号〉はポートの番号,そして〈式〉が出力するデータです.〈ポート番号〉は 0 ～32767までの整数で指定します.〈式〉は 0 から255までの整数で指定します.

**サンプルプログラム**

```
100 DEFINT A-Z
110 SCREEN 0,0:CLS 3:DIM PD(3):FOR I=0 TO 3:READ PD(I):NEXT
120 FOR X=80 TO 639 STEP 80
130  C=X/80
140  CIRCLE (X,100),50,C
150  PAINT (X,100),C,C
160 NEXT X
170 FOR C1=1 TO 24
180  FOR C2=0 TO 3
190   C=((C1+C2) MOD 8)*16+(C1+C2+4) MOD 8
200 '  OUT PD(C2),16*I+J は
210 '  COLOR=(C2,I):COLOR=(C2+4,J) と同じ
220   OUT PD(C2),C
230  NEXT C2
240  FOR W=0 TO 1000
250  NEXT W
260 NEXT C1
270 DATA &HAE,&HAA,&HAC,&HA8
280 END
```

# 〔1〕 PAINT

**機 能**　**指定された境界色で囲まれた領域を，指定された色で塗ります．**

**書 式**　PAINT ｜(Wx, Wy)｜ 〔, 〈領域色〉〔, 〈境界色〉〕〕
　　　　　　　　｜STEP(x, y)｜

**文 例**　PAINT (−50, 120), 3, 4

**解 説**

(Wx, Wy) により指定されたワールド座標を含む〈境界色〉で囲まれた領域を，指定された〈領域色〉で塗りつぶします．STEP を付けると相対座標となります．

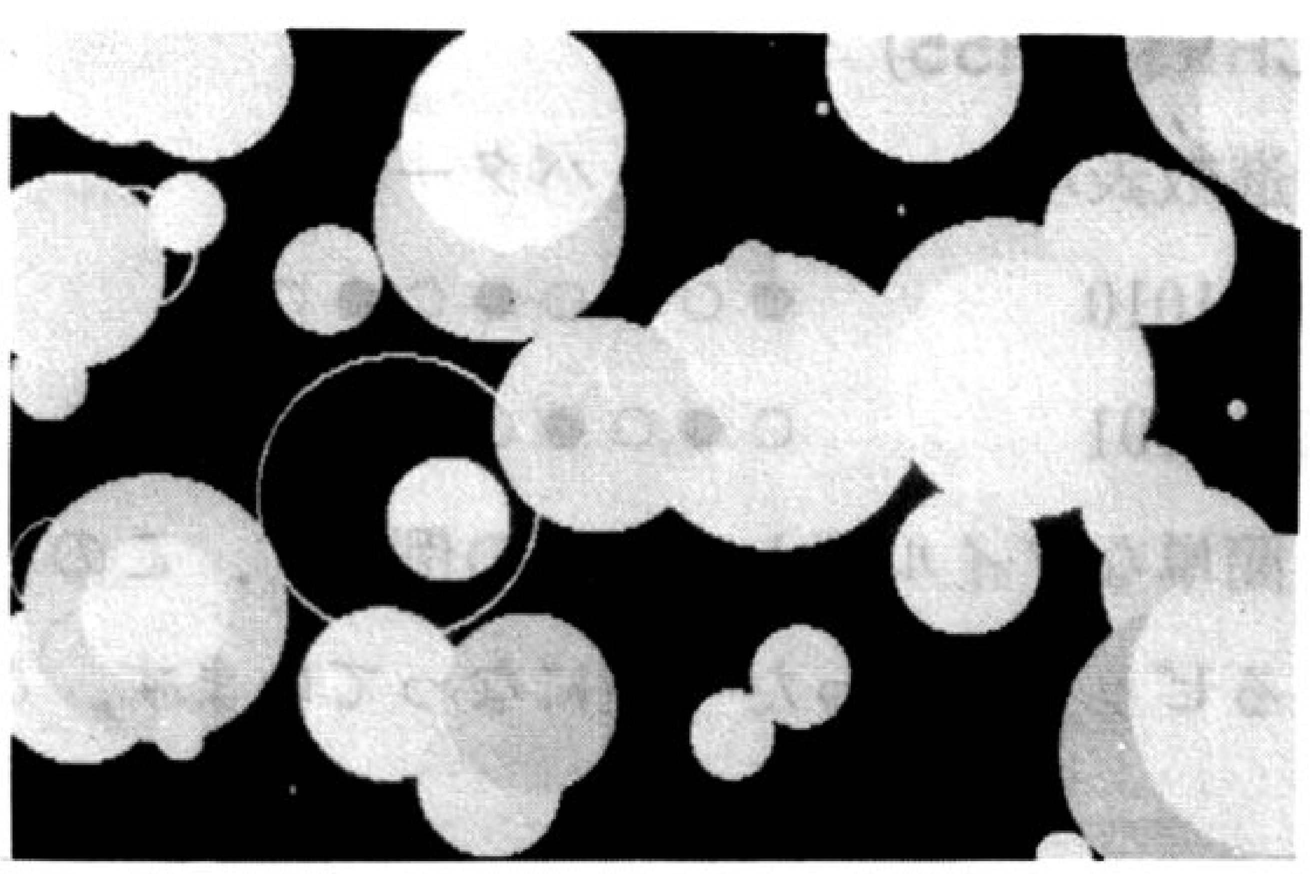
〈領域色〉，〈境界色〉はともにパレット番号で指定を行います．〈領域色〉が省略された場合には，COLOR 文で指定されたフォアグラウンドカラーが用いられます．〈境界色〉が省略された場合には〈領域色〉の指定と同じパレット番号が〈境界色〉として用いられます．

(Wx, Wy)は塗り始める座標を指定します．もしこの点が，既に指定された境界色と同じ色であった場合には，PAINT は何の画面操作も行いません．PAINT はビューポート内でのみ働きますので，ビューポートの境界は PAINT 動作の境界とみなされます．また，(Wx, Wy)がウィンドウの外にある場合には，対応するスクリーン座標はビューポート外になるために，"Illegal function call" エラーとなります．

参照：VIEW，〔1〕COLOR

**サンプルプログラム**

```
100 '-- normal paint --
110 SCREEN 0,0
120 CLS 3
130 FOR I=1 TO 50
140  X=RND*639
150  Y=RND*199
160  R=RND*80
170  C=INT(RND*7+1)
180  CIRCLE(X,Y),R,C
190  PAINT(X,Y),C
200 NEXT I
```



# 〔2〕 PAINT

**機　能**　**指定された境界色で囲まれた領域を，指定されたタイルパターンで埋めます．**

**書　式**　PAINT ｜(Wx, Wy)｜ , 〈タイルストリング〉〔, 〈境界色〉〕
　　　　　　　　｜STEP(x, y)｜

**文　例**　PAINT (255, 120), TILE$, 4

**解　説**

(Wx, Wy)により指定されたワールド座標を含む〈境界色〉で囲まれた領域を，〈タイルストリング〉により指定された模様(タイル)で埋めます．STEPを付けると相対座標による指定となります．

この動作はタイリングと呼ばれ，模様のついたタイルを領域内に敷きつめたり，中間色で領域内を塗ったような効果を出すことができます．

〈タイルストリング〉は，タイリングに用いられる基本タイルの模様と大きさを決定する文字列です．タイルの大きさは，横方向は8ドット分と決められていますが，縦方向の長さは，タイルストリングの長さで指定することができます．縦方向がnドットのタイルを指定するためには，白黒モードでn文字，カラーモードで3×n文字(4096色中・16色モードの場合は4×n文字)の長さを必要とします．

タイルの模様は，〈タイルストリング〉に対応するキャラクタコードの2進数表現（ビットパターン）により表現されます．〈タイルストリング〉の指定の方法はカラーモードと白黒モードとで異なります．白黒モードでは，文字を横8ビットの線に対応させて指定します．文字コードは，8ドットのうちのどのドットをセットし，どのドットをリセットにするかを決定し，2進数表現で1のビットに対応するドットはセット，0に対応するドットはリセットされます．〈タイルストリング〉がn文字の長さであれば，そのストリングの表す模様は，このようにして決定される横8ドットのパターンを縦にn文字のパターン分だけ並べたものになります．従ってタイルストリングを構成する文字が全て同じ文字であったり，1文字だけであった場合には，縦線模様になります．

例）CHR$(&HAA)＋CHR$(&H55)

| 16進数表現 | 2進数表現 | ドットパターン |
|---|---|---|
| &HAA | 10101010 | ●○●○●○●○ |
| &H55 | 01010101 | ○●○●○●○● |

上例は2文字からなる簡単なタイルストリングの例です．この2文字は互いに1であるビットと0であるビットがまったく逆になっています．このパターンを

基本タイルとして指定された領域を埋めれば，細かい市松模様になりますが，スクリーンのドットが大変細かいために，その模様はほとんどわからず，灰色のように見えるでしょう．

カラーモードでも白黒モードと同じ様に，模様はタイルストリングに対応するビットパターンによって決定されますが，白黒モードと異なり３文字(4096色中・16色モードの場合は４文字)で横８ドットが構成されます．ストリング中の文字はドットにどのパレット番号を対応させるかを決定します．ストリングの長さに余りがあった場合には無視されますが，３文字(あるいは４文字)に満たない場合には〝Illegal function call〞エラーとなります．

**例）　８色中・８色モードあるいは4096色中・８色モードの場合**

CHR$(&HAA)+CHR$(&H55)+CHR$(0)

| ドット | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| &HAA | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| &H55 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| &H00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 対応するパレット番号 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

８ドットにパレット番号１とパレット番号２を交互に指定します．パレットが初期状態であれば青と赤が交互に出る模様になります．しかし，ドットが細かいために，全体として紫色のようにみえます．

**例）　4096色中・16色モードの場合**

CHR$(&HCC)+CHR$(&H8F)+CHR$(&HD1)+CHR$(&H23)

| ドット | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| &HCC | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| &H8F | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| &HD1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| &H23 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 対応するパレット番号 | 7 | 5 | 8 | 4 | 3 | 3 | 10 | 14 |

**注意：**すでにタイリングされている領域を異なるパターンでタイリングしようとすると，時間がかかったり，スタックを消費するために〝Out of memory〞エラーになったりすることがあります．

サンプル
プログラム

```
100 '-- tiling paint (B/W mode) --
110 '
120 DATA 00,7F,63,55,49,55,63,7F
130 '
140 ' line 1 = 00000000 = &h00
150 ' line 2 = 01111111 = &h7f
160 ' line 3 = 01100011 = &h63
170 ' line 4 = 01010101 = &h55
180 ' line 5 = 01001001 = &h49
190 ' line 4 = 01010101 = &h55
200 ' line 3 = 01100011 = &h63
210 ' line 2 = 01111111 = &h7f
220 '
230 COLOR ,0:SCREEN 1,0,1,7:CLS 3
240 FOR I=1 TO 8
250  READ A$:TP=VAL("&H"+A$)
260  TILE$=TILE$+CHR$(TP)
270 NEXT I
280 CIRCLE(320,100),80,7
290 PAINT(320,100),TILE$,7
```

# PEN ON／OFF／STOP

**機 能**　**ライトペンによる割り込みの許可，禁止，停止を制御します．**

**書 式**　1)　PEN ON

2)　PEN OFF

3)　PEN STOP

**文 例**　PEN ON

**解 説**　ライトペンが押された時に起こる割り込みを許可（ON）するか，禁止（OFF）するか，停止（STOP）するかを設定します．

**書式 1）** 割り込みを許可します．以後ライトペンを押すごとに割り込みを発生し，ON PEN GOSUB 文で定義されている処理ルーチンに分岐します．

**書式 2）** 割り込みを禁止します．以後ライトペンを押しても処理ルーチンへの分岐は起こりません．

**書式 3）** 割り込みを停止します．以後ライトペンを押しても，そのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかしその後 PEN ON 命令によって割り込みが許可されると，先程のライトペンを押したことによって処理ルーチンに分岐します．

**注意：** プログラム終了時には PEN OFF を実行して割り込みを禁止しておいてください．

**参照：** ON PEN GOSUB，PEN

# POINT

機 能　Last Referenced Point (LP)を変更します.

書 式　POINT | (Wx, Wy) / STEP(x, y) |

文 例　POINT(−200, 30)

解 説　システムに記憶されている，最後にグラフィック操作の行われた座標を変更します(この座標値を Last Referenced Point (LP)といいます). この座標は，相対座標により座標指定を行う際の基点となりますが POINT 文はこの座標を設定する働きをします. したがって次の例1)と例2)とはまったく同じ意味を持つことになります.

例1)　LINE (−200, 30)−(100, 120), 3

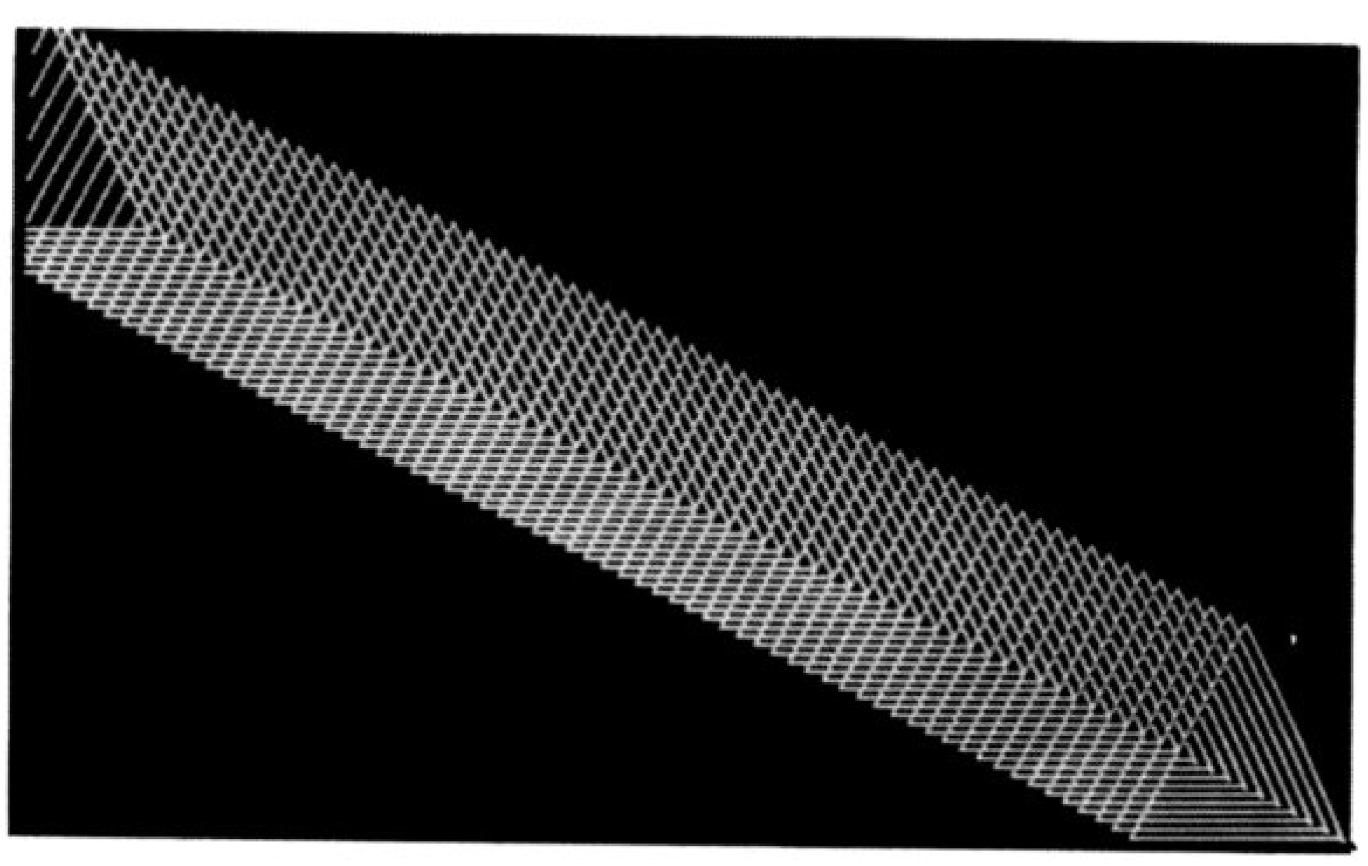
例2)　POINT (−200, 30)
　　　LINE −STEP (300, 90), 3

STEP を付けると相対座標による指定となります.

参照：LINE

サンプルプログラム

```
100 SCREEN 0,0:CLS 3
110 FOR I=0 TO 70
120  POINT STEP(8,2)
130  GOSUB *TRIANGLE
140 NEXT I:END
150 *TRIANGLE
160  LINE -STEP(-50,50),4
170  LINE -STEP(100,0),2
180  LINE -STEP(-50,-50),1:RETURN
```



# POKE

**機能**　メモリ上の指定番地へデータを書き込みます．

**書式**　POKE 〈番地〉，〈整数表記〉

**文例**　POKE &HF000，&HFF

**解説**

指定されたメモリ上の番地に1バイト（8ビット）のデータを書き込みます．

〈整数表記〉は書き込まれるデータで，0～255（&H0～&HFF）の値でなければなりません．もし小数点以下があると整数化してから実行されます．

〈番地〉は2バイトの値で，0～65535（&H0～&HFFFF）の範囲で指定します．この値はDEF SEG文で宣言されたセグメントベースからの相対アドレスを意味します．

この命令は，現在のメモリの内容を書き換えてしまうため，不用意に使うとBASICが使っている作業領域を壊してしまい，誤動作の原因となることもあります．使う時には，メモリマップなどで使用可能な領域かどうかを確認してください．

参照：PEEK，DFF SEG，VARPTR

# PRESET

| | |
|---|---|
| 機　能 | 画面上の任意のドットを消去します. |
| 書　式 | PRESET { (Wx, Wy) \| STEP(x, y) } 〔, <パレット番号>〕 |
| 文　例 | PRESET (100, 100) |

**解　説**

ワールド座標の(Wx, Wy)で表されるドットを消去します. STEPを付けた場合は相対座標による指定となります.

この命令は, オプションの<パレット番号>を付けない方が一般的な使い方で, その場合, 現在COLOR文によって設定されているバッググラウンドカラーで点を塗り変えます.

<パレット番号>を指定した場合は, PSET文とまったく同じに機能し, そのパレット番号のカラーでドットを表示します. PRESETはLP(Last Referenced Point)を指定された座標(Wx, Wy)に設定します.

LPに関する詳細はPOINT命令を参照してください.

参照：PSET, 〔1〕COLOR

サンプルプログラム

```
100 SCREEN 0,0:COLOR ,0:CLS 2
110 POINT(0,100)
120 FOR I=0 TO 200
130  PSET STEP(3,0),I MOD 8:NEXT I
140 POINT(0,100)
150 FOR I=0 TO 200
160  PRESET STEP(3,0)
170 NEXT I
```

# PRINT／LPRINT

機　能　　画面またはプリンタに情報を出力します.

書　式　　1) PRINT〔<式>…〕

　　　　　2) LPRINT〔<式>…〕

文　例　　PRINT "I="; I, "J="; J

解　説

　指定した式の値や文字列を，書式1)は画面に，書式2)はプリンタに出力します．<式>が省略されている場合は改行のみを行います．

　式の値や文字列を表示する領域は，あらかじめ各行を14文字毎に分割して定められており，区切り記号にコンマを使うと次の領域の始めから，またセミコロンを使うと直前にプリントしたもののすぐ後ろに続いて次の値や文字列が表示されます．区切り記号として空白を用いた場合もセミコロンと同様の働きをします．

　<式>の最後にセミコロン及びコンマを指定すると改行動作をおこさず，その行で次の PRINT 文による出力を続行します．

　変数と"(ダブルクォーテーション) で囲まれた文字列との区切りに限って",", ";", 空白を省略することができ，このときはセミコロンと同様の働きをします．

　数値を表示した場合その後ろに必ず空白が1つ挿入されます．また，数の前に符号のための桁を確保します．単精度の数値で，指数形式でなくても6桁以下の桁数で精度に影響を及ぼさず表示できるものは，実数形式で表示されます．

　同様に倍精度の数値は16桁以下の桁数で精度に影響を及ぼさず表示できるものは，実数形式の表示になります．

　この命令のキーワード"PRINT"の簡略形として疑問符(?)を使用することが許されます．

　表示する数値の長さが現在のカーソルの位置より後方にとれない場合(それを表示すると次の行にわたってしまう場合)は改行してからそれを表示します．

サンプルプログラム

```
100 FOR I=2 TO 15
110  FOR J=0 TO 15
120   PRINT CHR$(I*16+J);SPC(2);
130  NEXT J
140  PRINT
150 NEXT I
160 END

run
   !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
@  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  [  ¥  ]  ^  _
`  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
p  q  r  s  t  u  v  w  x  y  z  {  ¦  }  ~
▁  ▂  ▃  ▄  ▅  ▆  ▇  █  ▏  ▎  ▍  ▌  ▋  ▊  ▉  ┼
┴  ┬  ┤  ├  ▔  ─  │  ▕  ┌  ┐  └  ┘  ╭  ╮  ╰  ╯
   。 「 」 、 ・ ヲ ァ ィ ゥ ェ ォ ャ ュ ョ ッ
ー ア イ ウ エ オ カ キ ク ケ コ サ シ ス セ ソ
タ チ ツ テ ト ナ ニ ヌ ネ ノ ハ ヒ フ ヘ ホ マ
ミ ム メ モ ヤ ユ ヨ ラ リ ル レ ロ ワ ン ゛ ゜
═  ╞  ╪  ╡  ◢  ◣  ◥  ◤  ♠  ♥  ♦  ♣  ●  ○  ╱  ╲
×  円 年 月 日 時 分 秒
Ok
```

# PRINT #

**機能**　シーケンシャルファイルにデータを出力します．

**書式**　PRINT #〈ファイル番号〉，〔〈式〉…〕

**文例**　PRINT #2，A，B，C

PRINT #1，A$；〝,〟；B$

**解説**　〈ファイル番号〉は，そのシーケンシャルファイルが出力モードで開かれたときに指定された番号です．

〈式〉はファイルに書き込まれる数値式または文字式です．

PRINT #文はデータの圧縮を行わず，PRINT 文で画面へ出力するものと全く同じものを出力します．ですからこれらのデータがディスクから正しく入力できるように，ディスクに書き込まれたデータは適切に区切られるよう注意してください．

〈式〉中の数値式はセミコロンで区切るようにしてください．例えば，

```
A＝123
B＝456
C＝−78
PRINT #1, A ; B ; C
```

この場合ディスクに書込まれるデータは次のとおりです．

□123□□456□−78

(□は空白を意味します)

もし区切り記号にコンマを使うとプリント領域の間に挿入される余分な空白もディスクに書き込んでしまいます．挿入される空白の個数は PRINT 文の場合と同じです．

文字式もセミコロンで区切ってください．ディスク上に文字表記を正しく書き込むためには，区切り記号をデータとして出力させる必要があります．文字列の区切り記号はコンマとキャリッジリターン（CHR$(13)）です（詳細は INPUT #命令の説明を参照してください）．

例をあげると，A$＝〝CAMERA〟そして，B$＝〝93604−1〟のとき次の文，

```
PRINT #1, A$ ; B$
```

はディスクに CAMERA93604—1と書き込みます．これは区切り記号がありませんから２つの別の文字列として読み込むことはできません．この問題を解決するには，次のように PRINT #文中に独立に区切り記号を入れてください．

```
PRINT #1, A$; ″,″; B$
```

これによってディスクに書き込まれるイメージは,

```
CAMERA, 93604—1
```

となり, 2つの文字変数として読み込むことができます.

もし文字列それ自身がデータとしてコンマ, セミコロン, 意味のあるはじめの空白, キャリッジリターン, またはラインフィードなどを含む場合は, ダブルクォーテーション, CHR$(34)によって囲んでディスクに書いてください.

例をあげると, A$=″CAMERA, □AUTOMATIC″また, B$=″□□□93604—1″のとき（□は空白を意味します), 次の文,

```
PRINT #1, A$; B$
```

は次のようなイメージをディスクに書きます.

```
CAMERA, □AUTOMATIC□□□93604—1
```

そして次の文,

```
INPUT #1, A$, B$
```

は, A$ には ″CAMERA″ を, そして B$ には ″AUTOMATIC□□□93604—1″ を読み込んでしまいます. これらの文字列をディスクで正しく分割するには, CHR$(34)により引用符をディスクに書き込んでください. 次の文,

```
PRINT #1, CHR$(34); A$; CHR$(34); CHR$(34); B$; CHR$(34)
```

は次の内容をディスクに書き込みます.

```
″CAMERA, □AUTOMATIC″ ″□□□93604—1″
```

すると次の文,

```
INPUT #1, A$, B$
```

は ″CAMERA, □AUTOMATIC″ を A$に, ″□□□93604—1″ を B$に読み込みます.

PRINT #文は, USING と共に使ってディスクファイルのフォーマットを制御することができます.

なお, WRITE #文を使用すると適当にデータが区切られて書き込まれます.

**注意**：出力対象ファイルが″SCRN：″の場合, 日本語データの出力はできません.

**参照**：WRITE #, PRINT # USING, INPUT #

サンプル
プログラム

```
100 '-- Sequential data OUTPUT/INPUT --
110 '
120  OPEN "STEST.DAT" FOR OUTPUT AS #1
130  PRINT #1,DATE$;",";TIME$
140 *DATAWRITE
150  INPUT "品名";NA$
160  IF NA$="END" OR NA$="end" THEN *EXIT1
170  INPUT "価格";PLC
180  INPUT "個数";N%
190  PRINT #1,NA$;",";PLC;N%
200  GOTO *DATAWRITE
210 *EXIT1
220  PRINT:CLOSE
230 '-- Sequential data read --
240  TOTAL=0
250  OPEN "STEST.DAT" FOR INPUT AS #1
260  INPUT #1,DA$,TI$
270  PRINT "日付   :   ";DA$,"時間   :   ";TI$
280  PRINT
290 *DATAREAD
300   IF EOF(1) THEN *EXIT2
310   INPUT #1,NA$,PLC,N%
320   SUM=PLC*N%
330   PRINT NA$;TAB(10);PLC;"*";N%;"=";SUM
340   TOTAL=TOTAL+SUM
350  GOTO *DATAREAD
360 *EXIT2
370  PRINT:PRINT "TOTAL=";TOTAL
380 END
```

```
run
品名? 米
価格? 2500
個数? 1
品名? 牛肉
価格? 800
個数? 4
品名? END

日付  :   85/06/06          時間  :   06:06:06

米          2500 * 1 = 2500
牛肉        800 * 4 = 3200

TOTAL= 5700
Ok
```

# PRINT USING／LPRINT USING

---

**機能**　文字列，数値を編集出力します．

**書式**　| PRINT  | USING〈書式制御文字列〉;〈式〉〔| ; | 〈式〉…〕〔| ; |〕
　　　　　| LPRINT |　　　　　　　　　　　　　　　　　　　| , |　　　　　　　| , |

**文例**　PRINT USING "#####,."; A, B

**解説**　〈書式制御文字列〉によって後に続く〈式〉の出力される領域や書式を決定します．

**文字変数や文字列の書式制御**

- ！……与えられた文字変数や文字定数の最初の1文字だけ出力します．
- &〈n個の□(空白)〉&……与えられた文字変数や文字定数の先頭から(n+2)文字の文字列を出力します．与えられた文字列が(n+2)文字より長い場合は，余分な文字は無視され，短い場合には，文字列は左づめに出力され，残った部分には空白が出力されます．
- @……文字列全体の出力に使います．複数個指定した場合，1つの"@"に対して〈式〉中の1つの文字列が，代入・出力されます．"@"の数が文字列の個数より多い場合，残りの"@"は無視されます．

**数値の書式制御**

- #……数値を出力する桁数を指定します．指定した桁数より数値の桁数が小さいときには，右づめで出力されます．
- ．……小数点の位置を指定します．小数点以下の部分で冗長となる桁には0が出力されます．
- ＋……〈書式制御文字列〉の最初または最後に付けた場合，数値の符号がそれぞれ前または後に出力されます．2個以上の"＋"を並べた場合には，余分は後述の制御文字以外の文字と同じ扱いになります．
- －……〈書式制御文字列〉の最後に付けた場合，数値が負の数の時に数値の後ろに"－"が出力されます．前に付けたり，2個以上並べた場合には後述の制御文字以外の文字と同じ扱いになります．
- ＊＊……〈書式制御文字列〉の先頭につけた場合，数値領域の左側に空白部分ができたとき，そこを"＊"で埋めて出力します．この"＊＊"は2桁分の領域を確保します．
- ¥¥……〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に"¥"

を出力します．〝¥¥″は2桁分の領域を確保しますが，このうち1桁分は〝¥″の出力領域として使われます．後述の指数形式の書式指定を行った場合には，正しく出力されない事があります．

● ＊＊¥……〈書式制御文字列〉の先頭につけた場合，上記の2つ(＊＊と¥¥)の両方の機能となります．〝＊＊¥″は3桁分の領域を確保しますが，このうち1桁分は〝¥″の出力領域として使われます．

● ，……桁数指定の〝#″の並びの中においた場合，数値の整数部が3桁毎に〝,″で区切られて出力されます．ただし，上記の〝.″より右側においた場合は，数値の最後に，〝,″が出力され，3桁毎の区切りは行われません．

● ∧∧∧∧……桁数指定の〝#″の後に付けた場合，指数形式で出力されます．

● _……上記の制御文字1文字を単に文字として表示するために使用します．〝_″に続く1文字は常に書式制御機能を持たない文字として出力されます．

**制御文字以外の文字**

以上の制御文字以外の文字を置いた場合，数値の前や後ろにそのキャラクタが出力されます．ただし日本語文字は指定できません．

**数値の領域をこえた場合**

指定した数値領域より数値の桁数が大きい場合，数値の直前に〝%″が出力されます．数値の丸めが領域より大きくなる原因となった場合も，丸めた数値の前に〝%″が出力されます．

**注意**：日本語を含む文字列を編集することはできません．

サンプルプログラム

```
100 PRINT USING "!";"NEC COMPUTER"
110 PRINT USING "&        &";"NEC COMPUTER"
120 PRINT USING "#####";123.456
130 PRINT USING "#####.#";123.456
140 PRINT USING "+####.#";123.456
150 PRINT USING "#####.#-";-123.456
160 PRINT USING "**###.#";123.456
170 PRINT USING "¥¥###.#";123.456
180 PRINT USING "**¥##.#";123.456
190 PRINT USING "#####,.#";1234.56
200 PRINT USING "#####,.#^^^^";1234.56
210 PRINT USING "¥¥###._-";123.456
220 PRINT USING "#####";123456!
230 PRINT USING "This is a @.";"pen"
240 END
```

```
run
N
NEC COMPU
  123
  123.5
 +123.5
  123.5-
**123.5
 ¥123.5
*¥123.5
 1,234.6
 12345.6E-01
 ¥123.-
%123456
This is a pen.
Ok
```

# PRINT # USING

**機 能**　文字列，数値をファイルに編集出力します．

**書 式**　PRINT #<ファイル番号>，USING <書式制御文字列>；<式>〔 ; / , <式>…〕〔 ; / , 〕

**文 例**　PRINT #2，USING "####"；A

**解 説**　<ファイル番号>で指定されたファイルに対して，文字列や数値を編集して出力します．

PRINT # USINGは，その対象がファイルであることを除けばPRINT USINGとまったく同じです．

**注意**：日本語を含む文字列を編集出力することはできません．

**参照**：PRINT USING，PRINT #，OPEN

# PSET

機　能　　画面上にドットを表示します．

書　式　　PSET ｜(Wx, Wy)｜〔, 〈パレット番号〉〕
　　　　　　　　｜STEP(x, y)｜

文　例　　PSET(100, 100), 4

解　説

ワールド座標の(Wx, Wy)の位置にドットを表示します．STEP を付けると相対座標による指定となります．

カラーは〈パレット番号〉によって指定し，もし省略された場合は，現在のフォアグラウンドカラー(〔1〕COLOR 文参照)が採用されます．

PSET は LP(Last Referenced Point. 詳細は POINT 関数を参照してください)を指定された座標(Wx, Wy)に設定します．

参照：〔1〕COLOR, PRESET

サンプルプログラム

```
100 SCREEN 0,0:CLS 2
110 COLOR ,,,4
120 PSET(100,100)
130 '
140 POINT(150,100)
150 FOR I=0 TO 7
160  PSET STEP(10,0),I
170 NEXT I
```

# PUT

**機　能**　バッファ中のデータをファイルに書き出します.

**書　式**　PUT〔#〕<ファイル番号>〔,<数>〕

**文　例**　PUT #3, 5

**解　説**

〈ファイル番号〉で指定されたファイルに対応するバッファの内容を書き出します. PUT は, 指定されたファイルがディスクファイルであるかどうかで, その動作が異なります.

〈ファイル番号〉で指定されたファイルがディスクファイルの場合には, PUT は, ランダムファイルへの書き込みとして働きます. 指定されたファイルは, ランダムファイルのモードでオープンされていなければなりません. 〈数〉はファイルのレコード番号として解釈され, バッファ中のデータが指定されたレコードに書き込まれます. レコード番号が省略された場合には, 直前に行われた, GET, PUT で参照されたレコードの次のレコードに書き込まれます. レコード番号の最小値は 1, 最大値は65000です.

〈ファイル番号〉で指定されたファイルがプリンタあるいはスクリーンファイルの場合には, PUT は, バッファ中のデータを指定されたファイルに対して出力する働きをします. この時, 指定されたファイルは出力モードでオープンされていなければなりません. 〈数〉はバッファから, ファイルに対して書き出す文字数を意味し, ランダムバッファサイズの範囲内で指定します. 〈数〉が省略された時, 及び 0 が指定された時には, 256文字を書き出します.

なお書き出すデータは予め FIELD, LSET／RSET により準備しておかねばなりません.

参照：OPEN, FIELD, GET, LSET／RSET

サンプルプログラム

```
100 '-- random data write --
110 OPEN "DATA1" AS #1
120 FIELD #1,30 AS A$,20 AS B$,50 AS C$
130 *ENTRY
140 LINE INPUT "名前？";NAM$
150 IF NAM$="END" OR NAM$="end" THEN *EXIT
160 LINE INPUT "ＴＥＬ？";TEL$
170 LINE INPUT "住所？";ADR$
180 LSET A$=NAM$
190 LSET B$=TEL$
200 LSET C$=ADR$
210 PUT #1
220 GOTO *ENTRY
230 *EXIT
240 CLOSE
```

```
run
名前？岡野　徹
ＴＥＬ？０３－１１０－００００
住所？中央区築地７丁目
名前？松沢　謙治
ＴＥＬ？０４８８－１１－０１１９
住所？浦和市原山８丁目
名前？END
Ok
```

# PUT@

機 能　グラフィックパターンや漢字を画面に表示します．

書 式　1) PUT〔@〕(Sx, Sy), 〈配列変数名〉〔〈要素ナンバー〉〕, 〔, 〈条件〉〕〔, 〈フォアグラウンドカラー〉, 〈バックグラウンドカラー〉〕

2) PUT〔@〕(Sx, Sy), KANJI(〈漢字コード〉), 〔, 〈条件〉〕〔, 〈フォアグラウンドカラー〉, 〈バックグラウンドカラー〉〕

文 例　1) PUT(100, 100), G%, PSET

2) PUT(0, 0), KANJI(12321)

解 説　書式1)

GET@の命令によって配列に読み込まれたグラフィックパターンを画面上の任意の位置に表示します．

座標(Sx, Sy)はGET@の場合と同様で，ワールド座標ではなくスクリーン座標で指定します．またこの命令を実行するとLPは（Sx, Sy）に移動されます．

〈配列変数名〉は表示したいグラフィックパターンが格納されている配列名であり，〈要素ナンバー〉は配列内のどこからデータを取り始めるかの指定です．〈要素ナンバー〉が省略された場合は，配列の最初から取り始めます．これらの指定は，GET@命令で使ったものと同じものを指定します．

〈条件〉とは，グラフィックパターンを画面に表示する際，いろいろな機能を指定するもので，以下のものが用意されています．

PSET　配列内のグラフィックパターンをそのまま表示します．

PRESET　白黒モードの場合は配列内のパターンをリバースして表示します．カラーモードの場合は各ドットのパレット番号を，7―(そのドットのパレット番号)（4096色中・16色モードの場合は，15―(そのドットのパレット番号)）として表示します．

OR　配列内のグラフィックパターンと，既にある画面上のグラフィックパターンをビット（ドット）ごとにOR（論理和）し，その結果を画面に表示します．

AND　配列内のパターンと画面上のパターンをビットごとにAND（論理積）し，その結果を画面上に表示します．

XOR　配列内のパターンと画面上のパターンをビットごとにXOR（排他的論理和）し，その結果を画面に表示します．

※これらの〈条件〉は画面モードによって演算の対象が違います．白黒モードの場合，ドットがあるかないか（1ビット）を対象とし，カラーモードの場合，各パレット番号を対象とします．

**例）**（パレット3）AND（パレット6）→（パレット2）

もし〈条件〉が省略された場合は XOR とみなされます．

最後の〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉は，白黒モードの時に読み込んだパターンをカラーモードにおいて表示する時にのみ有効なオプションパラメータです．この2つのパラメータは両方とも指定するか，両方とも指定しないかのどちらかしか許されません．

〈フォアグラウンドカラー〉は，白黒モードで読み込んだ際に白であったドットに対しての色指定で，これをカラーモードにおいて実行すると任意の色に変えることができます．〈バックグラウンドカラー〉は同様に黒であったドットに対しての色指定です．これらは共にパレット番号によって指定します．

**注意**：GET@と PUT@命令は上記した〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉を指定して白黒モードで読み込んだパターンをカラーモードで表示する用途の他は，原則として同一画面モードで使用するようにしてください．

**書式2）**

〈漢字コード〉で指定された漢字または非漢字をグラフィック画面に表示します．

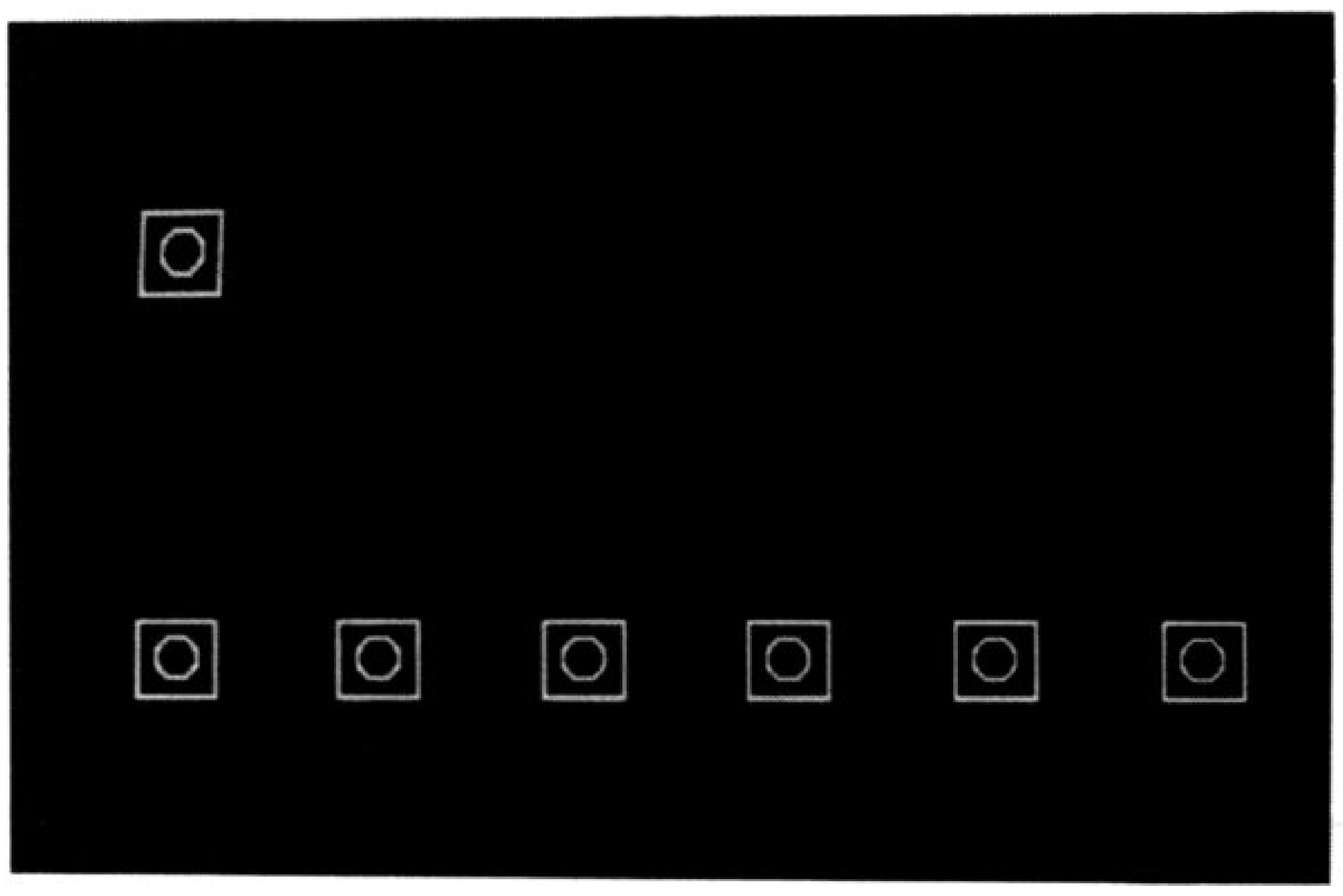
使用できる文字は，JIS 第1水準，第2水準および利用者定義文字で，これらは ROM および RAM の中に格納されています．ユーザーはこれらの中から任意の文字を〈漢字コード〉（ユーザーズマニュアル参照）によって指定し，画面上に日本語の文章を表示することができます．

書式1）の PUT 文ではユーザーが GET 文によって配列に読み込んだパターンを画面に表示しますが，漢字の PUT 文では配列内のデータにあたるものが，あらかじめ用意されています．このことを除けば，その使い方及び機能は書式1）の場合とほぼ同様です．ただし，漢字パターンは原則として縦16ドット，横16ドットの大きさと決められています（8×8，8×16のものもあります．ユーザーズマニュアル参照）．また漢字パターンは白黒モードで GET した場合と同じ形で格納されていますから，カラーモードにおいて〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉で指定してカラー表示させることも可能です．

**注意**：1つの PUT 文では1つの漢字しか表示することができません．

サンプル
プログラム

```
100 SCREEN 0,0:CLS 3
110 XD=80:YD=80
120 BYTE=((XD+7)¥8)*YD*4+4
130 FACT=BYTE¥2+1
140 DIM G%(FACT)
150 '
160 LINE(0,0)-STEP(XD-1,YD-1),1,B
170 CIRCLE(XD/2-1,YD/2-1),YD/4,4
180 GET(0,0)-STEP(XD-1,YD-1),G%
190 '
200 FOR X=0 TO 500 STEP 200
210 PUT(X,500),G%:NEXT
```



```
100 '-- kanji print --
110 DATA 427E,3A23,3441,3B7A,244E
120 DATA 2546,2539,2548,4366,212A
130 '
140 SCREEN 0,0:CLS 3
150 X=200
160 FOR I=1 TO 10
170  READ A$:KCODE=VAL("&H"+A$)
180  PUT(X,300),KANJI(KCODE),PSET,7,0
190  X=X+70
200 NEXT I
```

# RANDOMIZE

**機　能**　新しい乱数系列を設定します.

**書　式**　RANDOMIZE〔〈式〉〕

**文　例**　RANDOMIZE

**解　説**　新しい乱数の種(seed)を与えることによって乱数系列を変更します. 乱数のseedは〈式〉によって−32768から32767の間で与えます. 〈式〉が省略された場合, メッセージが出力されて, seedを要求してきます.

この場合も〈式〉と同じ範囲の値を入力しなければなりません. もしこの範囲を越えると〝Overflow〞エラーとなります.

参照：RND

**サンプルプログラム**

```
100 GOSUB *RNDSUB
110 CLEAR:GOSUB *RNDSUB
120 RANDOMIZE
130 GOSUB *RNDSUB
140 END
150 *RNDSUB'-- rnd number print
160  FOR I=0 TO 9
170   PRINT INT(RND*10);
180  NEXT I:PRINT
190 RETURN


run
 0  6  9  9  3  8  8  7  9  3
 0  6  9  9  3  8  8  7  9  3
Random number seed (-32768 to 32767)? 653
 2  1  6  5  7  8  3  6  4  3
Ok
```

# READ

| | |
|---|---|
| 機　能 | DATA 文より値を読み，変数に割り当てます. |
| 書　式 | READ <変数リスト> |
| 文　例 | READ K, M, S$ |

解　説

READ 文は DATA 文と組み合わせて使わなければなりません．READ 文は DATA 文のデータを 1 対 1 対応の方法で変数に割りあてていきます．READ 文の変数は数値変数でも文字変数でもかまいません．しかし読み出された値と変数の型は一致していなければなりません．もし，型が一致しない場合には，〝Syntax error〞が起こります．

1 つの READ 文が 1 つまたはそれ以上の DATA 文を参照したり(複数の場合は順番に参照されます)，またいくつかの READ 文が 1 つの DATA 文を参照することができます．もし<変数リスト>の変数の数が DATA 文のデータの個数を越えてしまった場合は，〝Out of DATA〞のエラーメッセージが表示されます．指定された変数が DATA 文のデータの個数よりも少ない場合には，その次の READ 文が読まれなかったデータから読み始めます．もしそれ以上 READ 文がない場合には，余分のデータは無視されます．

始めから，あるいは途中から DATA 文を読みなおすには，RESTORE 文を使います．

参照：RESTORE，DATA

サンプルプログラム

```
100 FOR I=0 TO 10
110  READ A
120  PRINT A;
130 NEXT I
140 READ A$:PRINT A$
150 END
160 DATA 10,9,8,7,6,5,4,3,2,1,0,Fire!

run
 10  9  8  7  6  5  4  3  2  1  0 Fire!
Ok
```

# REEM

|機 能| 注釈行を定義します.

|書 式| REM〔<注釈文>〕

|文 例| REM 給与計算

|解 説| REM文は非実行文であり，プログラムの実行に影響を与えずコメント行とすることができます．リストを取ると入力した内容がそのまま出力されます．

キーワード"REM"の代わりにアポストロフィ(')を使うことができます．REM文の行の後には，コロンで区切って他の文を続けることはできません．

# RENUM

| | |
|---|---|
| 機 能 | **プログラムに新しい行番号を付与します.** |
| 書 式 | RENUM〔<**新行番号**>〕〔, <**旧行番号**>〕〔, <**増分**>〕 |
| 文 例 | RENUM |

解 説

<新行番号>は，新しくつける行番号の最初の行番号で，省略値は10です．<旧行番号>は行番号のつけ替えをはじめる現在のプログラムの行番号です．省略値はそのプログラムの最初の行番号です．<増分>は新しく付ける各行番号のあいだの増分で，省略値は10です．

RENUM コマンドは，また，GOTO，GOSUB，THEN，ON…GOTO，ON…GOSUB および ERL 文などで参照している行番号も新しい行番号に対応して変更します．もしこれらの文が参照している行番号の行が存在しない場合には，〝Undefined line xxxx in yyyy〞とエラーメッセージが表示されます．この場合，誤った行番号 xxxx は RENUM コマンドによって変更されませんが，行番号 yyyy は変更されます．

**注意**：65529以上の行番号は発生することができません．このような場合には〝Illegal function call〞エラーが起こります．

R

# RESTORE

**機　能**　READ 文で読む DATA 文を指定します.

**書　式**　RESTORE〔〈行番号〉〕

**文　例**　RESTORE 800

**解　説**　〈行番号〉が省略されると，次にくる READ 文はプログラム中の最初の DATA 文から読み始めます．〈行番号〉を指定すると，指定された行の DATA 文から読み始めます．

**サンプルプログラム**

```
100  READ A,B,C
110 RESTORE
120  READ D,E,F
130 RESTORE *STRDATA
140  READ A$,B$,C$
150  PRINT A,B,C
160  PRINT D,E,F
170  PRINT A$,B$,C$
180 END
190  DATA 1,2,3
200  DATA 4,5,6
210 *STRDATA
220  DATA AA,BB,CC

run
 1              2              3
 1              2              3
AA             BB             CC
Ok
```

# RESUME

**機能** エラー回復処理終了後，プログラムの実行を再開します．

**書式** RESUME ｜〔0〕／NEXT／〈行番号〉｜

**文例** RESUME 0

RESUME NEXT

RESUME 1000

**解説** プログラムの実行を再開する場所に応じて次のように3つの書式を選ぶことができます．

| | |
|---|---|
| RESUME<br>または<br>RESUME 0 | エラーの原因となった文からプログラムの実行が再開されます． |
| RESUME NEXT | エラーの原因となった文のすぐ次の文から実行が再開されます． |
| RESUME 〈行番号〉 | 〈行番号〉で指定した所から実行が再開されます． |

参照：ON ERROR GOTO

**サンプルプログラム**

```
100  ON ERROR GOTO *ERRSUB
110 *ENTRY
120  INPUT "X,Y";X,Y
130  A=X/Y
140  PRINT "X/Y =";A
150  GOTO *ENTRY
160 *ERRSUB
170  PRINT "＊＊  計算できません  ＊＊"
180  RESUME *ENTRY


run
X,Y? 100,10
X/Y = 10
X,Y? 35,0
＊＊  計算できません  ＊＊
X,Y? 35,5
X/Y = 7
X,Y?
Break in 120
Ok
```

# ROLL

**機能**　**グラフィック画面を上下あるいは左右にスクロールさせます.**

**書式**　ROLL〔〈上下方向ドット数〉〕〔,〈左右方向ドット数〉〕〔, {N | Y}〕

**文例**　ROLL 16

**解説**

グラフィック画面を指定されたドット数だけ上下左右にスクロールします.〈上下方向ドット数〉とは画面の縦ドット数です. 許される値は640×200の分解能では－199から199, 640×400の分解能では－399から399までの値で, 正の場合には上方向に, 負の場合には, 下方向に指定のドット数(絶対値)だけスクロールします. 省略した場合, 上下方向のスクロールは行いません.

〈左右方向ドット数〉とは画面の横のドット数です. 許される値は－639から639までの値で, 正の場合には左方向に, 負の場合には右方向に, 指定のドット数(絶対値)以下で最も大きい 8 の倍数と等しいドット数だけ, スクロールします. 省略した場合, 左右方向のスクロールは行いません.

N または Y の指定は, スクロールにより新たに表れた領域をパレット番号 0 でクリアするか, バックグラウンドカラーでクリアするかを指定します.

Y を指定すると, バックグラウンドカラーでクリアし, N を指定するか, 省略したときには, パレット番号 0 でクリアします.

**サンプルプログラム**

```
100 SCREEN 1,0:CLS 3
110 FOR I=&H3000 TO &H5000 STEP &H100
120  FOR J=&H21 TO &H7E
130   KCODE=I+J
140   PUT(X,168),KANJI(KCODE),PSET
150   X=X+20
160   IF X>623 THEN X=0:ROLL 18
170  NEXT J
180 NEXT I
```

# RUN

**機能**　メモリにあるプログラムの実行を開始します．また，ディスクからプログラムをメモリにロードし，そのプログラムを実行します．

**書式**　1)　RUN〔<行番号>〕

2)　RUN <ファイルディスクリプタ>〔, R〕

**文例**　1)　RUN 100

2)　RUN "2 : TEST"

**解説**　**書式1)** <行番号>を指定すると，その行から実行がはじまります．指定のない場合には，最も若い行番号の行から実行がはじまります．プログラムの実行が終るとBASICはいつでもコマンドレベルに戻ります．

**書式2)** <ファイルディスクリプタ>で指定されるディスク上のプログラムをロード後実行を開始します．

RUNコマンドを実行すると，すべての開いているファイルは閉じ，目的のプログラムをロードする前にメモリの内容をクリアします．しかし，Rオプションを付けた場合には，すべてのデータファイルは開いたままになります．

**参照**：CHAIN

# SAVE

**機　能**　メモリにあるBASICプログラムをファイルにセーブします.

**書　式**　SAVE 〈ファイルディスクリプタ〉〔, | A | 〕
| P |

**文　例**　SAVE "2 : MYPROG", A

**解　説**

〈ファイルディスクリプタ〉で指定されるファイルにプログラムを書き込みます. 指定したファイル名と同じ名前のファイルが存在した場合には, 古い内容は失なわれて, 新しいものに更新されます.

Aオプションが指定された場合には, プログラムはアスキー形式でセーブされます(プログラムをLIST命令で表示した時と同じイメージでセーブが行われます). オプションの指定がない場合には, バイナリ形式に圧縮されてプログラムのセーブが行われます(メモリに格納されている時のイメージでそのままでセーブされます). アスキー形式のセーブは, バイナリ形式よりも, 多くのファイルスペースを必要としますが, コマンドによってはプログラムがアスキー形式でセーブされていることが条件となる場合があります. 例えばMERGEコマンドは, アスキー形式のファイルを必要とします. また, アスキー形式でセーブされたファイルは, データファイルとして読み出すことができます.

Pオプションを指定した場合には, プログラムは暗号化されたバイナリ形式でセーブされます. Pオプションでセーブされたプログラムをメモリにロードしても, LISTやEDITにより内容を見たり, 変更しようとすると, "Illegal function call"エラーとなります. 一度Pオプションにより保護が指定されると, これを解除する方法は用意されておらず, 二度と内容の変更等はできなくなるので注意してください.

参照 : LOAD, MERGE

# SCREEN

**機　能**　グラフィック画面に対して種々のモードを設定します．

**書　式**　SCREEN〔〈画面モード〉〕〔，〈画面スイッチ〉〕〔，〈アクティブページ〉〕
〔，〈ディスプレイページ〉〕

**文　例**　SCREEN 1，0，0，7

**解　説**　カラーモードにするか白黒モードにするか，カラーまたは白黒モードでどのページを書き込み及び表示ページとするかなど，ディスプレイのグラフィック画面に対しての種々のモードを設定します．

〈画面モード〉は，カラー，白黒，分解能など，グラフィック画面の最も基本的なモードを設定するパラメータで次の値をとります．

0――カラーモード(640×200ドット，4ページ)
1――白黒モード(640×200ドット，12ページ(16色モードの場合は16ページ))
2――高分解能白黒モード
(640×400ドット，6ページ(16色モードの場合は8ページ))
3――高分解能カラーモード(640×400ドット，2ページ)

〈画面スイッチ〉は，1ビットで表現される，〝グラフィックマスクスイッチ〟と〝高速書き込みスイッチ〟の2つのスイッチを合わせ持ったパラメータです．〝グラフィックマスクスイッチ〟をONにすると，現在グラフィック画面に描かれているパターンなどは一時的に消去されます．OFFにすれば元に戻ります．

〝高速書き込みスイッチ〟は特に意味のないダミースイッチです．

この2つのパラメータは次のように2ビットのビット対応になっており，与える値としては次のように0から3までとなります．

ビット1　ビット0
グラフィックマスクスイッチ→□□←高速書き込みスイッチ

0 ⟶ [0|0]　高速書き込み，グラフィックマスクスイッチ共OFF.
1 ⟶ [0|1]　高速書き込みスイッチのみON.

2 ⟶ 10　グラフィックマスクスイッチのみON.

3 ⟶ 11　高速書き込み，グラフィックマスクスイッチ共ON.

〈アクティページ〉と〈ディスプレイページ〉は書き込むページと表示ページの選択を行うためのものです.

〈アクティブページ〉と〈ディスプレイページ〉に指定できる値とその意味は8色モード（パレットモードが8色中・8色モードの場合あるいは4096色中・8色モードの場合）と16色モード（パレットモードが4096色中・16色モードの場合）で異なります（パレットモードに関しては〔1〕COLORを参照してください）.

### (1)　8色中・8色モードあるいは4096色中・8色モードの場合

〈アクティブページ〉には今後グラフィック命令によって書き込むページを0から11で指定します.

〈アクティブページ〉指定値とページ番号の関係

| ページ番号 | 画面モードごとのアクティブ・ページ指定値 | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 2 | 2 | 2 | × |
| 4 | 3 | 3 | 3 | × |
| 5 | × | 4 | 4 | × |
| 6 | × | 5 | 5 | × |
| 7 | × | 6 | × | × |
| 8 | × | 7 | × | × |
| 9 | × | 8 | × | × |
| 10 | × | 9 | × | × |
| 11 | × | 10 | × | × |
| 12 | × | 11 | × | × |

×印：指定不可

〈ディスプレイページ〉は画面モードに対応したページのうちどのページを表示するかを指定します。

〈ディスプレイページ〉指定値とその意味

| ディスプレイページの値 | 画面モードごとの意味 | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 1 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 |
| 2 | ページ2のみ表示 | ページ2のみ表示 | ページ2のみ表示 | × |
| 3 | × | ページ1,2を合成表示 | ページ1,2を合成表示 | × |
| 4 | × | ページ3のみ表示 | ページ3のみ表示 | × |
| 5 | × | ページ1,3を合成表示 | ページ1,3を合成表示 | × |
| 6 | × | ページ2,3を合成表示 | ページ2,3を合成表示 | × |
| 7 | × | ページ1,2,3を合成表示 | ページ1,2,3を合成表示 | × |
| 8 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 9 | × | ページ4のみ表示 | × | × |
| 10 | × | ページ5のみ表示 | × | × |
| 11 | × | ページ4,5を合成表示 | × | × |
| 12 | × | ページ6のみ表示 | × | × |
| 13 | × | ページ4,6を合成表示 | × | × |
| 14 | × | ページ5,6を合成表示 | × | × |
| 15 | × | ページ4,5,6を合成表示 | × | × |
| 16 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 17 | ページ3のみ表示 | ページ7のみ表示 | ページ4のみ表示 | ページ2のみ表示 |
| 18 | ページ4のみ表示 | ページ8のみ表示 | ページ5のみ表示 | × |
| 19 | × | ページ7,8を合成表示 | ページ4,5を合成表示 | × |
| 20 | × | ページ9のみ表示 | ページ6のみ表示 | × |
| 21 | × | ページ7,9を合成表示 | ページ4,6を合成表示 | × |
| 22 | × | ページ8,9を合成表示 | ページ5,6を合成表示 | × |
| 23 | × | ページ7,8,9を合成表示 | ページ4,5,6を合成表示 | × |
| 24 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 25 | × | ページ10のみ表示 | × | × |
| 26 | × | ページ11のみ表示 | × | × |
| 27 | × | ページ10,11を合成表示 | × | × |
| 28 | × | ページ12のみ表示 | × | × |
| 29 | × | ページ10,12を合成表示 | × | × |
| 30 | × | ページ11,12を合成表示 | × | × |
| 31 | × | ページ10,11,12を合成表示 | × | × |

×印：指定不可

### (2) 4096色中・16色モードの場合

〈アクティブページ〉には今後グラフィック命令によって書き込むページを0から15で指定します.

〈アクティブページ〉指定値とページ番号の関係

| ページ番号 | 画面モードごとのアクティブ・ページ指定値 | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 2 | 2 | 2 | × |
| 4 | 3 | 3 | 3 | × |
| 5 | × | 4 | 4 | × |
| 6 | × | 5 | 5 | × |
| 7 | × | 6 | 6 | × |
| 8 | × | 7 | 7 | × |
| 9 | × | 8 | × | × |
| 10 | × | 9 | × | × |
| 11 | × | 10 | × | × |
| 12 | × | 11 | × | × |
| 13 | × | 12 | × | × |
| 14 | × | 13 | × | × |
| 15 | × | 14 | × | × |
| 16 | × | 15 | × | × |

〈ディスプレイページ〉は画面モードに対応したページのうちどのページを表示するかを指定します．

| ディスプレイページの値 | 画面モードごとの意味 | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 1 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 |
| 2 | ページ2のみ表示 | ページ2のみ表示 | ページ2のみ表示 | × |
| 3 | × | ページ1，2を合成表示 | ページ1，2を合成表示 | × |
| 4 | × | ページ3のみ表示 | ページ3のみ表示 | × |
| 5 | × | ページ1，3を合成表示 | ページ1，3を合成表示 | × |
| 6 | × | ページ2，3を合成表示 | ページ2，3を合成表示 | × |
| 7 | × | ページ1，2，3を合成表示 | ページ1，2，3を合成表示 | × |
| 8 | × | ページ4のみ表示 | ページ4のみ表示 | × |
| 9 | × | ページ1，4を合成表示 | ページ1，4を合成表示 | × |
| 10 | × | ページ2，4を合成表示 | ページ2，4を合成表示 | × |
| 11 | × | ページ1，2，4を合成表示 | ページ1，2，4を合成表示 | × |
| 12 | × | ページ3，4を合成表示 | ページ3，4を合成表示 | × |
| 13 | × | ページ1，3，4を合成表示 | ページ1，3，4を合成表示 | × |
| 14 | × | ページ2，3，4を合成表示 | ページ2，3，4を合成表示 | × |
| 15 | × | ページ1，2，3，4を合成表示 | ページ1，2，3，4を合成表示 | × |
| 16 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 17 | × | ページ5のみ表示 | × | × |
| 18 | × | ページ6のみ表示 | × | × |
| 19 | × | ページ5，6を合成表示 | × | × |
| 20 | × | ページ7のみ表示 | × | × |
| 21 | × | ページ5，7を合成表示 | × | × |
| 22 | × | ページ6，7を合成表示 | × | × |
| 23 | × | ページ5，6，7を合成表示 | × | × |
| 24 | × | ページ8のみ表示 | × | × |
| 25 | × | ページ5，8を合成表示 | × | × |
| 26 | × | ページ6，8を合成表示 | × | × |
| 27 | × | ページ5，6，8を合成表示 | × | × |
| 28 | × | ページ7，8を合成表示 | × | × |
| 29 | × | ページ5，7，8を合成表示 | × | × |
| 30 | × | ページ6，7，8を合成表示 | × | × |
| 31 | × | ページ5，6，7，8を合成表示 | × | × |
| 32 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 33 | ページ3のみ表示 | ページ9のみ表示 | ページ5のみ表示 | ページ2のみ表示 |
| 34 | ページ4のみ表示 | ページ10のみ表示 | ページ6のみ表示 | × |
| 35 | × | ページ9，10を合成表示 | ページ5，6を合成表示 | × |
| 36 | × | ページ11のみ表示 | ページ7のみ表示 | × |
| 37 | × | ページ9，11を合成表示 | ページ5，7を合成表示 | × |
| 38 | × | ページ10，11を合成表示 | ページ6，7を合成表示 | × |
| 39 | × | ページ9，10，11を合成表示 | ページ5，6，7を合成表示 | × |
| 40 | × | ページ12のみ表示 | ページ8のみ表示 | × |
| 41 | × | ページ9，12を合成表示 | ページ5，8を合成表示 | × |
| 42 | × | ページ10，12を合成表示 | ページ6，8を合成表示 | × |
| 43 | × | ページ9，10，12を合成表示 | ページ5，6，8を合成表示 | × |
| 44 | × | ページ11，12を合成表示 | ページ7，8を合成表示 | × |
| 45 | × | ページ9，11，12を合成表示 | ページ5，7，8を合成表示 | × |
| 46 | × | ページ10，11，12を合成表示 | ページ6，7，8を合成表示 | × |
| 47 | × | ページ9，10，11，12を合成表示 | ページ5，6，7，8を合成表示 | × |

| | | | | |
|---|---|---|---|---|
| 48 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 49 | × | ページ13のみ表示 | × | × |
| 50 | × | ページ14のみ表示 | × | × |
| 51 | × | ページ13, 14を合成表示 | × | × |
| 52 | × | ページ15のみ表示 | × | × |
| 53 | × | ページ13, 15を合成表示 | × | × |
| 54 | × | ページ14, 15を合成表示 | × | × |
| 55 | × | ページ13, 14, 15を合成表示 | × | × |
| 56 | × | ページ16のみ表示 | × | × |
| 57 | × | ページ13, 16を合成表示 | × | × |
| 58 | × | ページ14, 16を合成表示 | × | × |
| 59 | × | ページ13, 14, 16を合成表示 | × | × |
| 60 | × | ページ15, 16を合成表示 | × | × |
| 61 | × | ページ13, 15, 16を合成表示 | × | × |
| 62 | × | ページ14, 15, 16を合成表示 | × | × |
| 63 | × | ページ13, 14, 15, 16を合成表示 | × | × |

この〈アクティブページ〉と〈ディスプレイページ〉の設定は気を付けないと，間違ってページを設定したり，書き込んだはずなのに表示ページになっていないなどのため，思った通りに表示できないことがあります．特に，8色モードと16色モードによって指定できる値とその意味が異なることに注意してください．

SCREEN命令を実行すると現在設定されているウィンドウ，ビューポート，LPは初期状態に戻されます（LP（Last Referenced Point）の初期状態はワールド座標，スクリーン座標とも（0，0）です）．

**注意**：白黒モード（高分解能白黒モードを含む）において，グラフィック命令で〈パレット番号〉を指定した場合は，0か0以外かによって黒か白かを判定します．また〈パレット番号〉を省略した場合は，カラーモードと同様，現在COLOR文によって設定されている〈フォアグラウンドカラー〉のパレット番号が採用されます．その際も0か0以外の判定により黒，白を決定します．

サンプルプログラム

### 8色モードの例

```
screen 0,0,0,1   'カラー(200),ページ 1 に書いてページ 1 を表示
screen 0,0,2,17  'カラー(200),ページ 3 に書いてページ 3 を表示
screen 3,0,0,1   'カラー(400),ページ 1 に書いてページ 1 を表示
screen 3,0,1,17  'カラー(400),ページ 2 に書いてページ 2 を表示
screen 2,0,0,1   '白黒(400),ページ 1 に書いてページ 1 を表示
screen 2,0,1,2   '白黒(400),ページ 2 に書いてページ 2 を表示
screen 2,0,2,7   '白黒(200),ページ 3 に書いてページ 1,2,3 を合成表示
screen 1,0,3,9   '白黒(200),ページ 4 に書いてページ 4 を表示
screen 1,0,4,10  '白黒(200),ページ 5 に書いてページ 5 を表示
screen 1,0,5,15  '白黒(200),ページ 6 に書いてページ 4,5,6 を合成表示
```

# SET　(DISK モード)

**機能**　ファイル属性のセット・リセットを行います.

**書式**
1)　SET〈ドライブ番号〉,〈〝属性文字〞〉
2)　SET〈ファイル番号〉,〈〝属性文字〞〉
3)　SET #〈ファイル番号〉,〈〝属性文字〞〉

**文例**
SET 1, 〝P〞
SET 〝2 : NOTWRITE〞, 〝P〞
SET #1, 〝R〞

**解説**　指定したドライブ, ファイル番号, ファイルに,〈〝属性文字〞〉によって指定された属性を付けます. 属性文字は, P, R で指定し, それぞれ, ライトプロテクト (書き込み禁止), リードアフターライト (書き込み確認) の属性を付けます. これ以外の文字が属性文字として指定された場合には, 現在設定されている属性が全て解除されます.

**書式 1)**〈ドライブ番号〉が指定された場合には, 指定されたドライブのディスク媒体に対して属性がつけられます.

**書式 2)**〈ファイル名〉が指定された場合には, 指定されたファイルのみに属性がつけられ同一ディスク媒体中の他のファイルにはその影響は及びません.

**書式 3)**〈ファイル番号〉が指定された場合には, そのファイルが開かれている間だけ指定された属性が作用します.

P属性を付けると, PRINT #, PUT, WRITE #等の書き込み動作が禁止されるとともに, ファイルの KILL もできなくなります.

R 属性を付けると, 書き込みを行った直後に読み出しを行い, 正しく書き込みが行われたかの確認がされるようになります.

**参照**: ATTR$

# STOP

---

| | |
|---|---|
| 機　能 | **プログラムの実行を一時中断します.** |
| 書　式 | STOP |
| 文　例 | STOP |

解　説

STOP 文はプログラムの実行を一時中断するためのもので，プログラムのどこで使用してもかまいません．STOP 文を実行すると，次のメッセージが表示されます．

Break in <行番号>　　　(<行番号>はストップした行番号です)

END 文と異なり，STOP 文はファイルを閉じません．

STOP が実行されると，BASIC は常にコマンドレベルに戻ります．また，CONT コマンドによりプログラムの実行を再開することができます．INPUT 文などの入力文の途中でストップした場合，継続することはできません．

参照：CONT，END

# STOP ON／OFF／STOP

機能　STOP キーおよび CTRL-C キーによる割り込みの許可，禁止，停止を制御します．

書式
1) STOP ON
2) STOP OFF
3) STOP STOP

文例　STOP ON

解説
書式1）割り込みを許可します．以後 STOP キーあるいは CTRL-C キーが押されるごとに ON STOP GOSUB 文によって設定された処理ルーチンに分岐します．

書式2）割り込みを禁止します．以後 STOP キーあるいは CTRL-C キーを押しても割り込み処理ルーチンへの分岐は起こりません（この時 STOP キーあるいは CTRL-C は通常の動作をします）．

書式3）割り込みを停止します．以後 STOP キーあるいは CTRL-C キーを押してもその事を覚えているだけで，処理ルーチンへの分岐は起こりません．しかし STOP ON 命令によって割り込みが許可されると，先程の STOP キーあるいは CTRL-C キーを押したことによる割り込みで処理ルーチンに分岐します．

参照：ON STOP GOSUB，KEY ON／OFF／STOP

サンプルプログラム

```
100 ON STOP GOSUB *BREAK
110 STOP ON
120 GOTO 120
130 *BREAK
140  PRINT "BREAK
150  RETURN
```

# SWAP

| | |
|---|---|
| 機能 | **2つの変数の値を入れ替えます.** |
| 書式 | SWAP 〈変数〉, 〈変数〉 |
| 文例 | SWAP A$, B$ |

解説

どの型の変数でも，SWAP文によりその値を交換することができます(整数，単精度，倍精度，文字，または配列変数)．しかしその2つの変数の型は一致していないと〝Type mismatch〟エラーが起こります.

サンプルプログラム

```
100 ' 2つの配列の内容を入れ換える
110  DIM A(10),B(10)
120  FOR I=1 TO 10
130   A(I)=I*2:B(I)=I*3:GOSUB *PRINTAB
140  NEXT I:LOCATE 0,Y+3
150  FOR I=1 TO 10
160   SWAP A(I),B(I):GOSUB *PRINTAB
170  NEXT I:LOCATE 0,Y+2
180 END
190 *PRINTAB
200  X=POS(0):Y=CSRLIN
210  LOCATE X,Y:PRINT USING "###";A(I);
220  LOCATE X,Y+1:PRINT USING "###";B(I);
230  LOCATE X+5,Y
240 RETURN
```

```
run
  2    4    6    8   10   12   14   16   18   20
  3    6    9   12   15   18   21   24   27   30

  3    6    9   12   15   18   21   24   27   30
  2    4    6    8   10   12   14   16   18   20
Ok
```

# TERM

**機　能**　システムをターミナルモードにします.

**書　式**　TERM "〔COM : 〕〔〈パリティ〉〔〈ビット〉〔〈ストップビット〉〔〈XON スイッチ〉〔〈S パラメータ〉〔〈DEL コード処理〉〔〈RET キー処理〉〔〈受信$C_R$ コード処理〉〔〈日本語シフトコード指定〉〕〕〕〕〕〕〕〕〕〔, 〔〈モード〉〕〔, 〈リモートBASIC プロトコル処理用領域の大きさ〉〕〕

**文　例**　TERM "COM : E72X",, 1000

**解　説**

BASIC モードからターミナルモードに制御を移行します. ターミナルモードでは, RS-232C インターフェースを介して他のコンピュータや機器との通信が可能となります. また, ターミナルモード時においても, リモート BASIC プロトコルにより, $N_{88}$-BASIC の機能を利用することができます. 詳しくはユーザーズマニュアルを参照してください.

最初の COM : でコミュニケーションポートを使って外部機器との入出力を行うことを宣言します. なお, COM : は省略することができますが, 省略した場合は, 後に続くすべてのパラメータを指定しなければなりません.

〈パリティ〉は E, O, N で表わし, それぞれ偶数パリティ, 奇数パリティ, パリティ無しを意味します.

〈ビット〉は, 1 文字を表わすのに用いるビット数で, 7 または 8 で指定します. 7 を指定した場合には, 必ず〈パリティ〉で O または E を指定しなければなりません. また 8 を指定した場合には〈パリティ〉は N でなければなりません.

〈ストップビット〉は, ストップビット数を決定し, 1, 2, 3 で指定します. 1 は 1 ビット, 2 は1.5ビット, 3 は 2 ビットを意味します.

〈XON スイッチ〉は, XON／XOFF による通信制御を行うことを指定し, スイッチとして X が指定されると XON／XOFF 制御を行います. N が指定された場合にはこの制御は行われません.

〈S パラメータ〉は〈ビット〉に 7 を指定した時に, キャラクタコード128以上の文字（カナ文字など）の入出力ができなくなるのを補助するパラメータです. 指定は S または N で行い, S を指定した時入出力可能で, N を指定した時は入出力不可能になります.

〈DEL コード処理〉は, 受信した DEL コードの処理方法を決定するもので, B または N で指定します.

B が指定されると, BS コードに変換されて BS コードとして処理されます.

Nが指定されると，NULコードに変換されてNULコードとして処理されます．

〈RETキー処理〉は，RETURNキーを押したときに送信するコードを決定するもので，CまたはLで指定します．

Cが指定されると，$C_R$ コードを送信します．

Lが指定されると，$C_R$ コードとLFコードを送信します($C_R$，LFの順に送信します)．

〈受信$C_R$ コード処理〉は，$C_R$ コードを受信したときの動作を決定するもので，CまたはLで指定します．

Cが指定されると，1つの$C_R$ コードの受信で，復帰動作と改行動作が行われます．

Lが指定されると，$C_R$ コードの受信では，復帰動作だけが行われ，$C_R$ コードとLFコードを連続して受信されたときに，復帰動作と改行動作が行われます．

〈日本語シフトコード指定〉は，英数カナ文字列の中における日本語文字列の始まりを表示するコード（KIコード）と日本語文字列の終りを表示するコード（KOコード）を決定するもので，PまたはIで指定します．

Pが指定されると，KIコード＝$(1B4B)_{16}$〔ESC・Kに相当〕，
KOコード＝$(1B48)_{16}$〔ESC・Hに相当〕
が使用されます．

Iが指定されると，KIコード＝$(1A70)_{16}$〔SB・pに相当〕，
KOコード＝$(1A71)_{16}$〔SB・qに相当〕
が使用されます．

〈モード〉は，HまたはFで指定し，それぞれ半二重，全二重通信のモードを意味します．

〈リモートBASICプロトコル処理用領域の大きさ〉は，リモートBASICプロトコルを利用する際に用いられる変数領域の大きさを決定します．省略された場合1024に設定されます．

ターミナルモードからBASICモードに戻る時には，シフトキーとストップキーを同時に押します．

**注意**：〈変数領域の大きさ〉を除くすべてのパラメータは，省略された場合，リセットした時（電源投入時も同様）に設定されていたメモリスイッチの値が採用されます（メモリスイッチについては，ユーザーズマニュアルを参照してください）．また途中のパラメータを省略して，後のパラメータを指定する場合は，省略記号としてブランク（〝 〟）を指定しなければなりませ

ん.

ここで使用する各パラメータは，必ず大文字を使ってください.

**注意**：ターミナルモードを RS-232C 第 2 回線および第 3 回線に適用することはできません.

T

# TIME$ ON／OFF／STOP

| | |
|---|---|
| 機　能 | リアルタイムタイマによる割り込みの許可，禁止，停止を制御します. |
| 書　式 | 1） TIME$ ON<br>2） TIME$ OFF<br>3） TIME$ STOP |
| 文　例 | TIME$ ON |

解　説

書式１）割り込み動作を許可します．この命令実行後は，設定された時刻になると割り込みが発生し，ON TIME$ GOSUB 文によって定義された処理ルーチンに分岐します．

書式２）割り込み動作を禁止します．この命令実行後は，割り込みが発生しません．

書式３）割り込み動作を停止します．この命令実行後は，割り込みが発生したことを覚えているだけで，処理ルーチンの分岐は起こりません．しかし，TIME$ ON 命令により割り込みが許可されると，先程の割り込みによって処理ルーチンへ分岐します．

注意：プログラム終了時には TIME$ OFF を実行してください．

参照：ON TIME$ GOSUB，TIME$

サンプルプログラム

```
100 TIME$="07:25:00"
110 ON TIME$="07:30:00" GOSUB 140
120 TIME$ ON
130 GOTO 130
140 PRINT "TIME LIMIT"
150 RETURN 100
```

# TRON／TROFF

| 機　能 | プログラムの実行状態を追跡します. |
| 書　式 | TRON<br>TROFF |
| 文　例 | TRON<br>TROFF |

解　説

　プログラムのデバッグの助けとして，TRON 文を実行すると，トレースフラグが立ち，実行するプログラム行の行番号を表示します．それぞれの行番号は角カッコで囲まれ表示されます．トレースフラグは TROFF 文(または NEW コマンド)を実行するとリセットされます．

サンプルプログラム

```
100 FOR I=1 TO 3
110  PRINT I,
120 NEXT

TRON
Ok
Run
[100][110] 1   [120][110] 2   [120][110] 3   [120]
Ok
TROFF
Ok
Run
 1              2              3
Ok
```

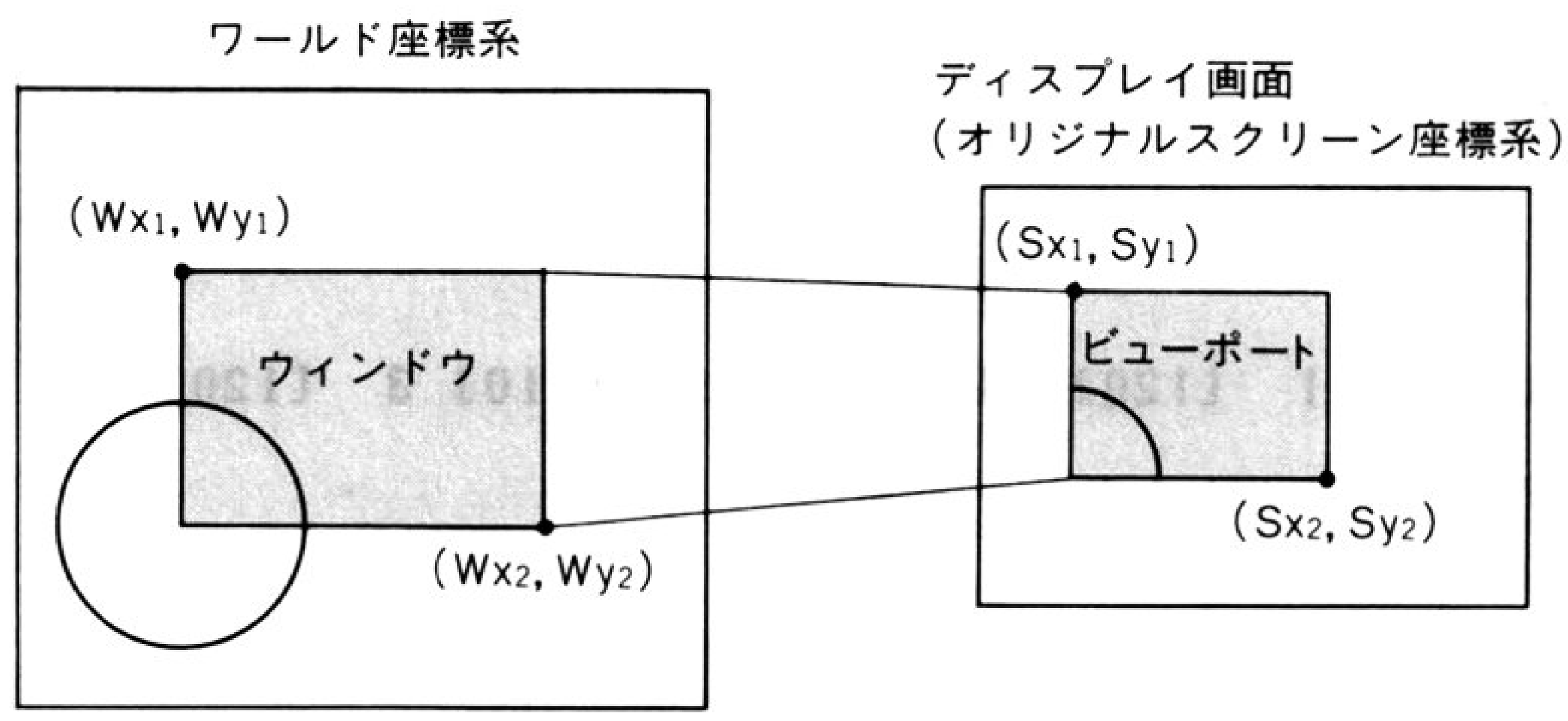
# VIEW

**機能**　**ディスプレイ画面上での表示領域(ビューポート)を指定します.**

**書式**　VIEW ($Sx_1$, $Sy_1$)−($Sx_2$, $Sy_2$) 〔, 〈**領域色**〉〕〔, 〈**境界色**〉〕

**文例**　VIEW (100, 30)−(200, 75),, 7

**解説**　($Sx_1$, $Sy_1$)を左上の頂点, ($Sx_2$, $Sy_2$)を右下の頂点とする, オリジナルスクリーン座標系上の長方形を図形表示領域として指定します. ここでいうオリジナルスクリーン座標とは, ビューポート内に展開されるスクリーン座標ではなく, ディスプレイ画面のドットと1対1に対応して, WINDOW, VIEW により変化することのない物理的な座標のことです. 以後, WINDOW により指定されているワールド座標系上の領域内に入る図形は, VIEW で指定された領域中に表示されるようになります. この領域はビューポート(View Port)と呼ばれます.




〈領域色〉が指定された場合には, 指定されたパレット番号でビューポート内を塗りつぶします.

〈境界色〉が指定された場合には, 指定されたパレット番号でビューポートの枠を描きます. CLS によって消去される画面の範囲は, この枠によって囲まれた中だけで, 枠は消されることはありません.

VIEW は, 図形の表示領域の指定を行うだけで, 実際の画面に対する操作は行いませんので, VIEW によりビューポートを変更するだけで, 以前のビューポート中の図形が移動するようなことはありません. また, VIEW はすべてのグラフィック画面表示の対象範囲を指定された領域に限定してしまいますから, ビューポートの外に点や線等を表示することはできなくなります.

VIEW の指定を変えるだけで, 1つの図形を描くルーチンでも, 画面上の異なる位置に異なる大きさで描くようになります.

$Sx_1 < Sx_2$, $Sy_1 < Sy_2$ が成り立たない場合，あるいはこれらの座標がディスプレイ画面から外れている場合には，"Illegal function call" エラーとなります．

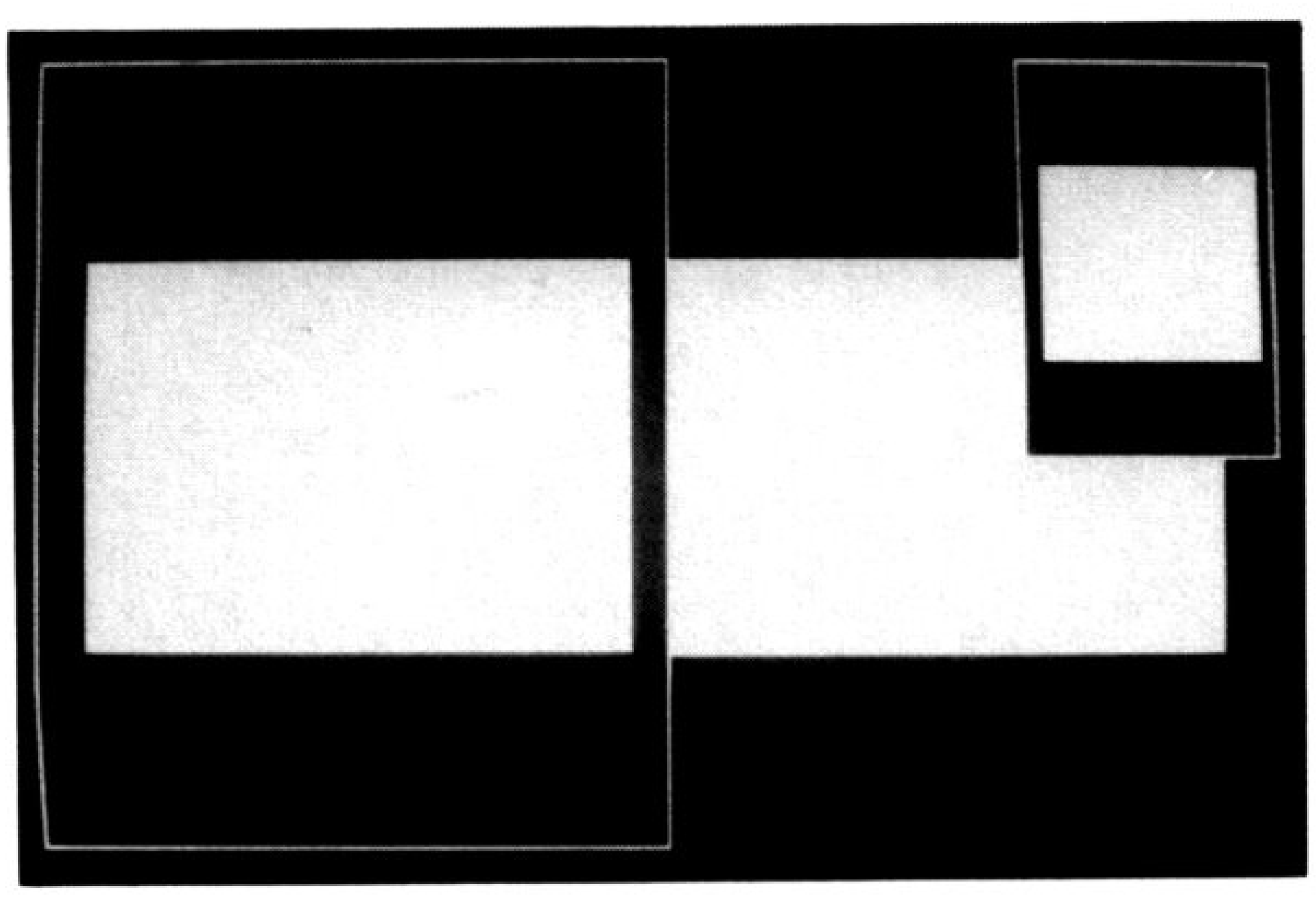
一度設定されたビューポートは，次に VIEW 文あるいは SCREEN 文が用いられるまで変化しません．また VIEW は LP をビューポートの左上の頂点に設定します．

**注意**：ワールド座標系がビューポート内に展開されるのは，WINDOW 文の実行後であり，VIEW のみ実行してウィンドウは初期状態のままであれば，ビューポート内の座標は，(ワールド座標)＝(スクリーン座標) となります．

**参照**：WINDOW，CLS，SCREEN

サンプルプログラム

```
100 SCREEN 0,0:CLS 2
110 WINDOW(0,0)-(639,199)
120 GOSUB *RECT
130 VIEW(1,1)-(319,198),0,7
140 GOSUB *RECT
150 VIEW(500,1)-(630,99),0,7
160 GOSUB *RECT
170 END
180 *RECT
190  LINE(50,50)-(600,150),7,BF
200 RETURN
```



# WAIT

機能　コンピュータの入力ポートをモニタする間，プログラムの実行を停止します．

書式　WAIT <ポート番号>，<式1>〔，<式2>〕

文例　WAIT 1，&H22，&H22

解説　WAIT 文を実行すると，指定した入力ポートのビットパターンが指定した状態になるまでプログラムの実行が中断されます．指定したポートから読み込んだデータと<式2>とのXORの結果と<式1>のANDが取られます．もしその結果が0(偽)なら，BASICはもう一度ポートの状態を読み込み同じ操作を繰り返します．もし結果が0でなければ(真)，プログラムの実行は次の文に移ります．<式2>が省略された場合は0と見なします．

注意：WAIT 文の実行により無限ループにはいってしまう場合があります．その場合にはコンピュータをリセットしなければなりません．

サンプルプログラム

```
100 '-- random from key port --
110 PRINT "Type 's' to start."
120 WAIT &HE4,8,255
130 '-- START --
140 END

run
Type 's' to start.
Ok
s
```

# WHILE～WEND

機　能　　条件が満足されている間，指定されている区間の実行を繰り返します．

書　式　　WHILE <論理式>
　　∫
WEND

文　例　　WHILE J<5
　　∫
WEND

解　説

<論理式>が真(0でない)である間，WHILE文とWEND文の間にある文の実行を繰り返します．

<論理式>を評価した結果が真であれば，WHILE文とWEND文との間にある文を順番に実行し，WHILE文に戻ります．この繰り返しは<論理式>の評価が偽になるまで続けられ，<論理式>が偽になると，WEND文に続く文に制御が移ります．最初に<論理式>を評価した結果が偽であれば，WHILE文とWEND文との間にある文は一度も実行されません．

WHILE～WEND文はFOR～NEXT文と同じ様に入れ子構造にすることができます．この場合には，それぞれのWEND文は，それ以前の最も近くにあるWHILE文と対応すると解釈されます．WEND文は，WHILE文と対になり，ループの終りを示す働きをしますので省略することはできません．

サンプルプログラム

```
100 I=1
110 WHILE I<6
120  PRINT I;"-";
130  J=1
140  WHILE J<=I
150   PRINT J;:J=J+1
160  WEND
170  PRINT:I=I+1
180 WEND
```

W

# WIDTH

**機　能**　各種入出力機器やファイルに対して１行の長さ等を指定します．

**書　式**
1）WIDTH <桁数>〔，<行数>〕
2）WIDTH <ファイルディスクリプタ>，<サイズ>
3）WIDTH <ファイル番号>，<サイズ>

**文　例**
1）WIDTH 80，25
2）WIDTH "LPT 1 :"，80
3）WIDTH #1，100

**解　説**

**書式１）** テキスト画面に表示する文字の行数を指定します．１行あたり40文字，また80文字の <桁数> を設定できます．また <行数> は20行か25行のどちらかです．<行数>を省略した場合は現在の行数のままで変化しません．

**書式２）** <ファイルディスクリプタ>で指定されたデバイスに対して１行の長さを指定します．ここで許されているデバイスは，コミュニケーションポートとプリンタです．<サイズ>の値は０から255まで許されており０は256と解釈されます．たとえば，この命令がプリンタに対して実行されたとすれば，プリンタの１行に印字される文字数は <サイズ> で指定したものとなります．このようにプリンタに対して実行した場合，WIDTH LPRINT と同等の機能を得ることができます．初期状態では255に設定されています．

**書式３）** <ファイル番号> で割り当てられているバッファ（チャネル）に対して，その大きさを <サイズ> で指定します．ここで許されているデバイスは，コミュニケーションポートとプリンタです．サイズは０から255までの値で０は256と解釈されます．以後このファイル番号を使っているデバイスとの入出力はこの <サイズ> 単位で行われます．初期状態では255に設定されています．

なお，コミュニケーションポートに対して WIDTH 命令を実行すると，コミュニケーションポートに対するデータの送出時，<サイズ>で指定された桁数の位置ごとに改行（CR）コードを送出することになります．

**参照**：WIDTH LPRINT

# WIDTH LPRINT

**機　能**　プリンタに出力される1行の出力文字数を設定します.

**書　式**　WIDTH LPRINT 〈文字数〉

**文　例**　WIDTH LPRINT 80

**解　説**　プリンタの1行に印字することのできる文字数を設定します.〈文字数〉で指定することのできる値は0から255までで, 0は256と解釈されます.

**注意**:日本語1文字は文字数2文字でカウントします.また,日本語文字列の前後には文字数で4文字分の制御コードが挿入されますのでこれも計算に入れなければなりません.

**参照**:WIDTH

サンプルプログラム

```
100 WIDTH "LPT:",255
110 WIDTH LPRINT 40
120 FOR I=33 TO 247
130  LPRINT CHR$(I);
140 NEXT I
150 LPRINT:END
```

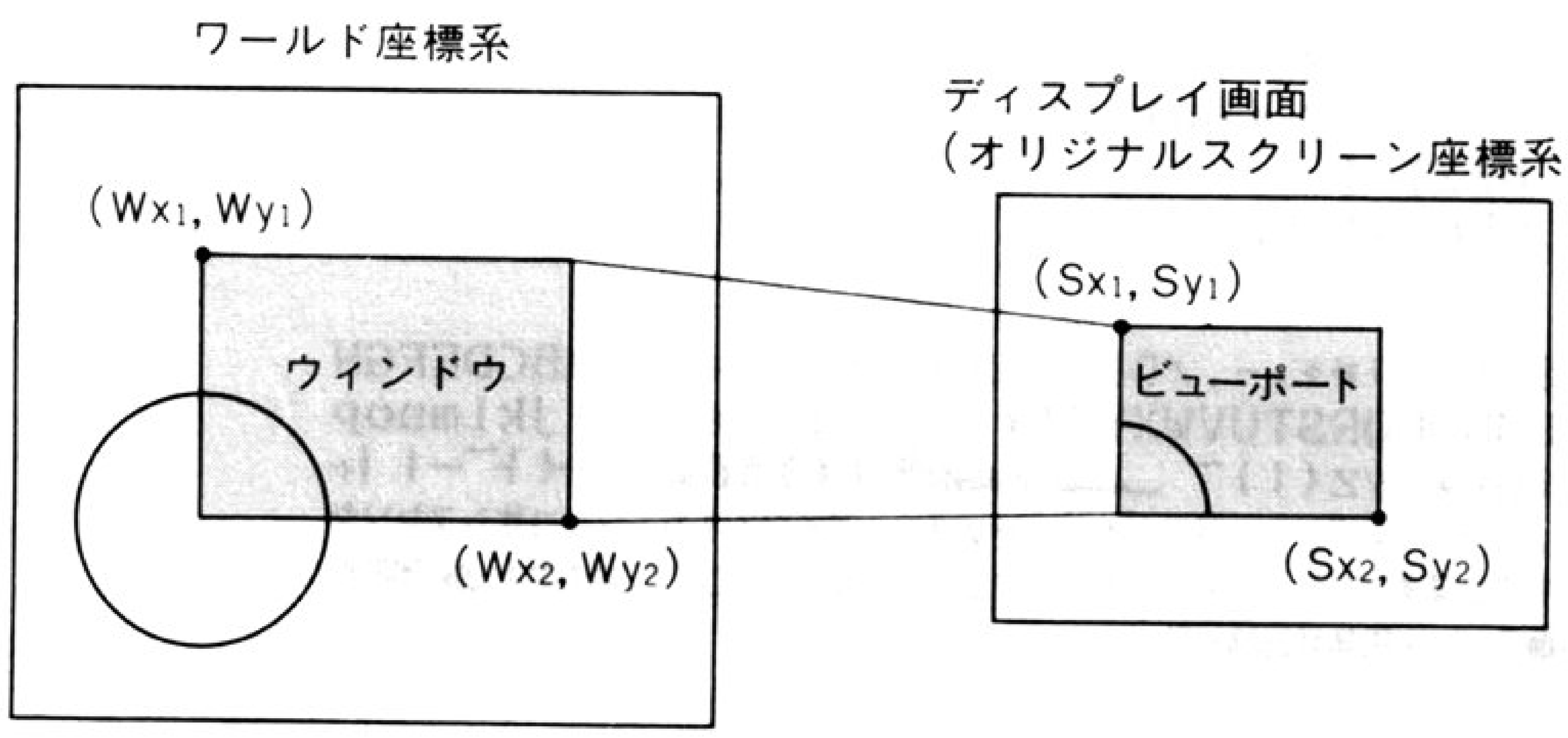
```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGH
IJKLMNOPQRSTUVWXYZ[¥]^_`abcdefghijklmnop
qrstuvwxyz{|}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔─│▕┌
┐└┘╭╮╰╯ 。「」、・ヲァィゥェォャュョッーアイウエオカキクケコサシスセソタ
チツテトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン゛゜═╪╡▲▼♠
♥♦♣●○╱╲╳円年月日時分秒
```

# WINDOW

| 機能 | ビューポートに表示するワールド座標系内の領域を指定します． |
|---|---|
| 書式 | WINDOW（$Wx_1$，$Wy_1$）－（$Wx_2$，$Wy_2$） |
| 文例 | WINDOW（－300，－50）－（100，70） |

解説

（$Wx_1$，$Wy_1$）を左上の頂点，（$Wx_2$，$Wy_2$）を右下の頂点とするワールド座標系上の長方形で囲まれた領域を，スクリーン上のビューポートに表示する領域として指定します．以後，この領域内に入る図形はVIEWで指定された領域（ビューポート）内に表示されるようになります．このようにして指定された領域は，ウィンドウと呼ばれ，ちょうどカメラのファインダを通して景色を見た時の様に，ウィンドウから外れた図形は表示されなくなります．また，ウィンドウの大きさを変えることにより，カメラのズーミングの様に，視野に入る領域の大きさが変り，それがスクリーン上のビューポートに表示されるようになります．




WINDOWもVIEWと同じ様に領域の指定を行うだけで，実際の画面に対する操作も行いませんので，WINDOWにより，ウィンドウの設定を変更するだけで，ビューポート中に表示される図形を変化するようなことはありません．また，誤って，図形が何も描かれない領域をウィンドウとして設定すると，スクリーン上のビューポートには，何の図形も描かれませんので注意してください．

$Wx_1 < Wx_2$，$Wy_1 < Wy_2$ が成り立たない場合，あるいはこれらの座標がワールド座標系から外れている場合には〝Illegal function call〞エラーとなります．WINDOWで指定する座標は，ワールド座標ですから，負の値や実数値もとれることに注意してください．

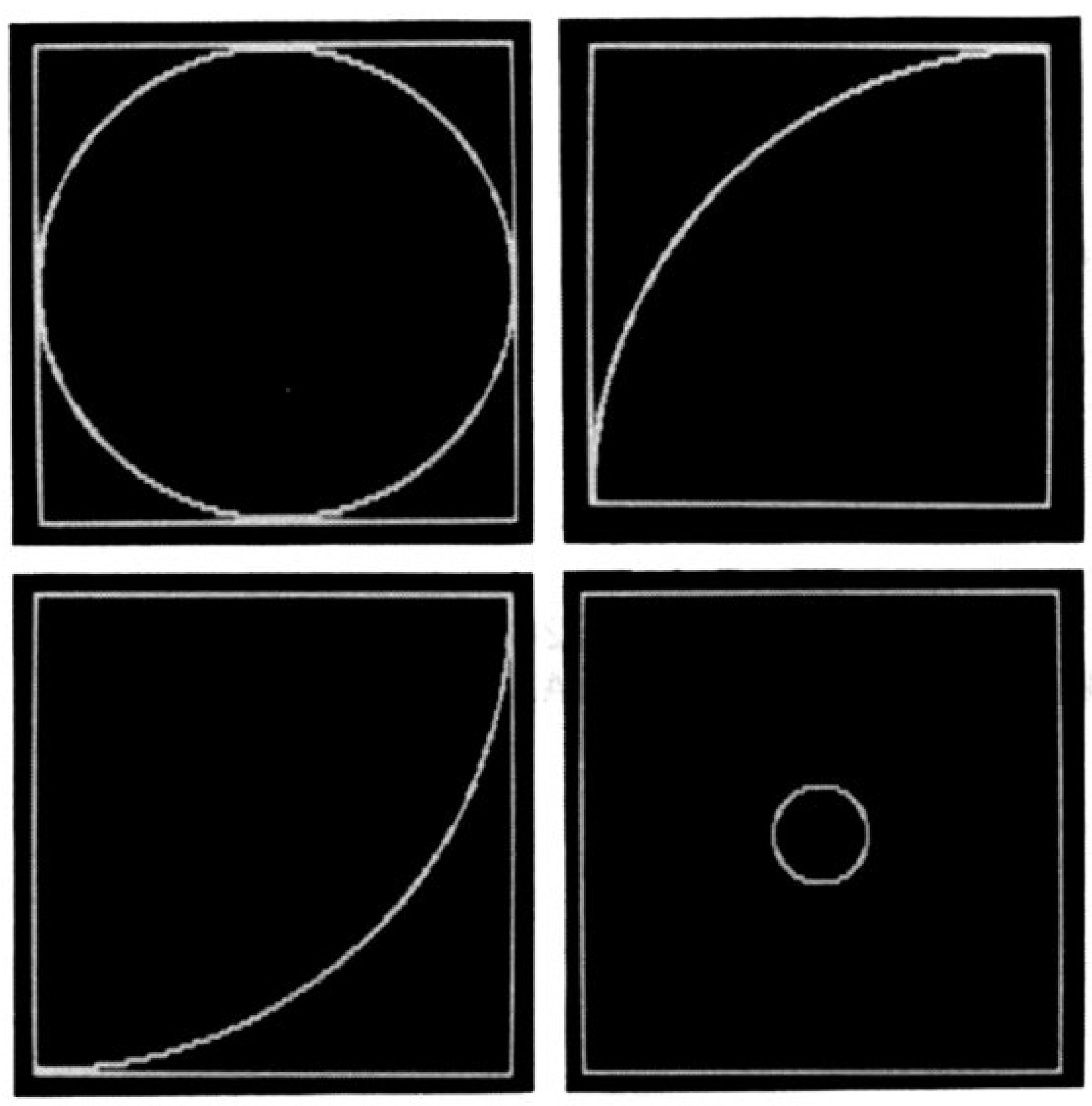
一度設定されたウィンドウは，次にWINDOW文あるいはSCREEN文が用いられるまで変化しません．またWINDOWはLPをウィンドウの左上の頂点に設定します．

**注意**：ワールド座標系がビューポート内に展開されるのは，WINDOW 文の実行後であり，VIEW のみ実行してウィンドウは初期状態のままであれば，ビューポート内の座標は，(ワールド座標)＝(スクリーン座標) となります．

**参照**：VIEW

サンプルプログラム

```
100 SCREEN 0,0:CLS 2
110 C=1
120 VIEW(200,50)-(400,150),,7
130 WINDOW(-100,-100)-(100,100)
140 GOSUB *CIRC
150 WINDOW(-100,-100)-(0,0)
160 GOSUB *CIRC
170 WINDOW(0,0)-(100,100)
180 GOSUB *CIRC
190 WINDOW(-500,-500)-(500,500)
200 GOSUB *CIRC
210 END
220 *CIRC
230  CIRCLE(0,0),100,C
240  FOR J=0 TO 1000:NEXT J
250  C=C+1:CLS 2
260 RETURN
```



# WRITE

**機　能**　画面へデータを出力します.

**書　式**　WRITE〔<式の列>〕

**文　例**　WRITE I ＊ J, K ; A$

**解　説**　〈式の列〉により指定された，数値式あるいは文字式の値を画面に出力します.〈式の列〉が省略された場合には空行のみが出力されます．式はコンマ(,)あるいはセミコロン(；)により区切ります．コンマとセミコロンとの機能上の区別はありませんが，画面上には，各式の値の区切りとして必ずコンマが表示されます．BASICは，各式の値を出力したのち，改行を行います．

WRITE文は，ほぼPRINT文と同じ様に式の値の出力を行いますが，不要な空白桁は詰められ，それぞれの式の間は必ずコンマで区切られます．また文字列はダブルクォーテーション(")で囲まれて出力されます．

参照：PRINT，WRITE #

**サンプルプログラム**

```
100 A%=123
110 B!=9.87654E+31
120 C#=3.14159265359#
130 D$="N88-BASIC"
140 PRINT A%;B!,C#;D$
150 WRITE A%;B!,C#;D$

run
 123  9.87654E+31              3.14159265359 N88-BASIC
123,9.87654E+31,3.14159265359,"N88-BASIC"
Ok
```

# WRITE #

**機能** シーケンシャルファイルへデータを書き出します.

**書式** WRITE #<ファイル番号>〔, <式の列>〕

**文例** WRITE #1, I * J, K ; A$

**解説** <式の列>により指定された数値式あるいは文字式の値を, <ファイル番号>により指定されたファイルに書き出します. 指定されたファイルは出力用にオープンされていなければなりません. <式の列>中の各式は, コンマ(,)あるいは, セミコロン(;)によって区切ることができます.

WRITE #は, ほぼPRINT #と同じように式の値を出力しますが, 不要な空白桁は詰められ, それぞれの式の値の間は必ずコンマで区切られます(コンマそのものがデータとして出力されます). また文字列はダブルクォーテーション(")で囲まれて出力されます.

WRITE #は, 各式の値を書き出した後, 改行のコードを書き出します. <式の列> が省略された場合には, 単に改行のコードのみが書き出されます.

WRITE #は, 区切り記号としてコンマを必ず出力し, また不要な空白の出力は行わないため, PRINT #に比べファイルの使用領域を節約することができます.

注意：出力対象ファイルが"SCRN:"の場合, 日本語データを出力することはできません.

参照：PRINT #, WRITE

**サンプルプログラム**

```
100 '-- write --
110 OPEN "DATA5" FOR OUTPUT AS #1
120 INPUT "名前   ？",NA$
130 INPUT "番号   ？",NM
140 INPUT "点数   ？",PO
150 WRITE #1,NA$,NM,PO
160 CLOSE

run
名前   ？松沢   謙治
番号   ？23
点数   ？99
Ok
```

# 第3章

# 関　数

# ABS



| | |
|---|---|
| 機　能 | **絶対値を与えます.** |
| 書　式 | ABS(<数式>) |
| 文　例 | B＝ABS(－2) |

**解　説**

〈数式〉の絶対値を返します.

また, 〈数式〉に倍精度実数が含まれる場合は, 倍精度の値を与えますが, 他の場合には, 単精度の値を与えます.

**サンプルプログラム**

```
100 '――  ２点間の距離を求める  ――
110 INPUT "点１の座標は：Ｘ, Ｙ";X1,Y1
120 INPUT "点２の座標は：Ｘ, Ｙ";X2,Y2
130 D=SQR(ABS(X1-X2)^2+ABS(Y1-Y2)^2)
140 PRINT:PRINT "点１と点２の距離は  ";D


run
点１の座標は：Ｘ, Ｙ? 100,500
点２の座標は：Ｘ, Ｙ? 20,64

点１と点２の距離は   443.279
Ok
```

# AKCNV$

| | |
|---|---|
| 機　能 | **文字列の中の1バイト系文字を2バイト系日本語文字に変換します.** |
| 書　式 | AKCNV$(<文字列>) |
| 文　例 | A$=AKCNV$(B$) |

解　説

　<文字列>の中の1バイト系文字を2バイト系日本語文字に変換します.

サンプルプログラム

```
100 A$="ｲﾛﾊ漢字ABC"
110 PRINT AKCNV$(A$)
```

```
run
イロハ漢字ＡＢＣ
Ok
```

# ASC

機　能　文字のキャラクタコードを与えます.

書　式　ASC(<文字列>)

文　例　A＝ASC("A")

解　説　<文字列>の最初の文字のキャラクタコードを与えます.

参照：CHR$

サンプルプログラム

```
100 '−− 小文字 −> 大文字変換 −−
110 INPUT A$
120 IF A$="" THEN 190
130 B$=""
140 FOR I=1 TO LEN(A$)
150  D$=MID$(A$,I,1):D=ASC(D$)
160   IF D<97 OR D>122 THEN B$=B$+D$:GOTO 180
170  B$=B$+CHR$(D-32)
180 NEXT I
190 PRINT:PRINT B$


run
? nec

NEC
Ok
```

# ATN

---

|機　能| 逆正接(アークタンジェント)を与えます.

|書　式| ATN(<数式>)

|文　例| A＝ATN(1)

|解　説| <数式>の逆正接値をラジアンで与えます．得られる値は，$-\pi/2$から$\pi/2$までの範囲です．

また，<数式>に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には，単精度の値を与えます．

|サンプルプログラム|

```
100 '-- ARCTAN から ARCSIN,ARCCOS を求める  ――
110 T=180/3.14159:U1$="#.###":U2$="##.####      ###.##"
120 INPUT A
130 IF ABS(A)>1 THEN 120
140 IF ABS(A)=1 THEN AS=1.5788:AC=0:GOTO 170
150 AS=ATN(A/SQR(-A*A+1))
160 AC=-ATN(A/SQR(-A*A+1))+3.14159/2
170 DS=AS*T:DC=AC*T
180 PRINT "                        Rad.       Deg."
190 PRINT "arcCOS(";USING U1$;A;:PRINT ") =";USING U2$;AC;DC
200 PRINT "arcSIN(";USING U1$;A;:PRINT ") =";USING U2$;AS;DS
210 END
```

```
run
? .5
                   Rad.       Deg.
arcCOS(0.500) = 1.0472      60.00
arcSIN(0.500) = 0.5236      30.00
Ok
```

# ATTR$　　(DISK モード)

**機　能**　**指定ファイル，ドライブの属性を与えます．**

**書　式**　ATTR$ ｜ (<ドライブ番号>) ｜ (# <ファイル番号>) ｜ (<ファイル名>) ｜

**文　例**　PRINT ATTR$(1)

**解　説**

指定したドライブ，オープンされているファイル，ディスク上のファイルの現在の属性文字を与えます．この時，属性文字は以下の意味を持ちます．

〝□□□″：属性は解除されています．即ち，通常のリード，ライトが可能な状態です．

〝R□□″：書き込みの際に，リードアフターライト(書き込んだ内容とメモリ上の内容の比較チェック)を行います．

〝□□P″：書き込みが禁止されています．

〝□E□″：Pオプションが指定されています．

参照：SET

# CDBL

| | |
|---|---|
| 機　能 | **整数値，単精度実数値を倍精度実数値に変換した数値を与えます．** |
| 書　式 | CDBL(<数式>) |
| 文　例 | A #＝CDBL(B!／2) |

解　説　<数式>の値を倍精度実数値に変換します．ただし，型変換が行われるだけで有効桁数の変化がありませんから，得られた値の精度は，変換する前の型と同じ(整数型なら整数部のみ，単精度実数型なら有効数字 6 桁)になります．

参照：CINT，CSNG

サンプルプログラム

```
100 '  誤 差 の な い 倍 精 度 実 数 へ の 変 換
110 A!=1.23456:GOSUB 150:GOSUB 210
120 A!=123456!:GOSUB 150:GOSUB 210
130 A!=1.23456E+08:GOSUB 150:GOSUB 210
140 END
150 A$=STR$(A):A1#=CDBL(A!)
160 D=INSTR(A$,".")
170 IF D=0 OR INSTR(A$,"E")<>0 THEN A2#=A1#:RETURN
180 D=LEN(A$)-D
190 A2#=INT(CDBL(A!*10^D+.5))/10^D
200 RETURN
210 PRINT A!,A1#;TAB(34);A2#
220 RETURN
```

```
run
 1.23456        1.234560012817383    1.234558649701477
 123456         123456               123456
 1.23456E+08    123456000            123456000
Ok
```

# CHR$

機　能　　指定したキャラクタコードを持つ文字を与えます.

書　式　　CHR$(<数式>)

文　例　　A$＝CHR$(65)

解　説

<数式>の値のキャラクタコードを持つ文字を与えます.値が0～255の範囲にない場合は,〝Illegal function call〞エラーになります.

参照：ASC

サンプルプログラム

```
100 '−−  大文字  −> 小文字変換  −−
110 INPUT A$
120 IF A$="" THEN 200
130 B$=""
140 FOR I=1 TO LEN(A$)
150  D$=MID$(A$,I,1):D=ASC(D$)
160  IF D<65 OR D>90 THEN B$=B$+D$:GOTO 180
170  B$=B$+CHR$(D+32)
180 NEXT I
190 PRINT:PRINT B$
200 END


run
? NEC PERSONAL COMPUTER

nec personal computer
Ok
```

# CINT

| | |
|---|---|
| 機　能 | **単精度実数値，倍精度実数値を整数値に変換した値を与えます．** |
| 書　式 | CINT(〈数式〉) |
| 文　例 | A%=CINT(B#*2) |

解　説

〈数式〉の値の小数点以下を四捨五入して整数に変換した値を与えます．結果の値が−32768〜32767の範囲にない場合は，〝Overflow〞エラーになります．

参照：CSNG，CDBL

サンプルプログラム

```
100 '  小数点以下の任意の桁での四捨五入
110 A=1.23456:D=2:GOSUB 140:GOSUB 170
120 A=1.23456:D=3:GOSUB 140:GOSUB 170
130 END
140 IF A=INT(A) THEN AD=A:RETURN
150 AD=CINT(A*10^D)/10^D
160 RETURN
170 PRINT A,AD
180 RETURN


run
 1.23456        1.23
 1.23456        1.235
Ok
```

# COS



| | |
|---|---|
| 機 能 | **余弦(コサイン)を与えます.** |
| 書 式 | COS(<数式>) |
| 文 例 | C=COS(3.1415／2) |

解 説

<数式>の値に対する余弦値を与えます. <数式>の単位はラジアンです.

また, <数式>に倍精度実数が含まれる場合は, 倍精度の値を与えますが, 他の場合には, 単精度を与えます.

参照：SIN, TAN

サンプルプログラム

```
100 '  ＣＯＳからＳＩＮを求める
110 INPUT "角度（度）  :  ";D
120 CS=COS(D/180*3.1415962#)
130 PRINT "COS";USING "###";D;:PRINT " = ";CS,
140 PRINT "SEC";USING "###";D;
150 IF ABS(CS)<=.000001 THEN 180 ELSE SC=1/CS
160 PRINT " = ";SC
170 END
180 PRINT " = 定義できません"
190 END
```

```
run
角度（度）  :  ? 45
COS 45 =  .707106          SEC 45 =  1.41421
Ok
```

# CSNG

| | |
|---|---|
| 機　能 | **整数値，倍精度実数値を単精度実数値に変換した値を与えます．** |
| 書　式 | CSNG(〈数式〉) |
| 文　例 | A!=CSNG(B#) |

解　説

〈数式〉の値を有効数字6桁の単精度実数型に変換します．

その値が−1.70141E+38〜1.70141E+38の範囲にない場合は，〝Overflow〞エラーになります．

参照：CINT，CDBL

サンプルプログラム

```
100 '  小数点以下の任意の桁での四捨五入
110 A#=1.23456789#:D=1:GOSUB 140:GOSUB 170
120 A#=1.23456789#:D=7:GOSUB 140:GOSUB 170
130 END
140 IF A#=INT(A#) THEN AD#=A#:RETURN
150 AD#=INT(A#*10^D+.5)/10^D
160 RETURN
170 PRINT A#,CSNG(AD#)
180 RETURN

run
 1.23456789    1.2
 1.23456789    1.23457
Ok
```

# CSRLIN

| | |
|---|---|
| 機　能 | **現在のカーソルの行位置を与えます．** |
| 書　式 | CSRLIN |
| 文　例 | Y=CSRLIN |

解　説

現在のカーソルの行位置を与えます．値の範囲は25行モードでは 0 ～24，20行モードでは 0 ～19となります．なお，画面の最上行が 0 ，最下行が24(または19)となっています(POS 関数参照)．

参照：POS

サンプルプログラム

```
100 CONSOLE 0,25,0,0
110 '  ビリヤードのシミュレーション
120 LOCATE 1,2:PRINT STRING$(78,"●");
130 LOCATE 1,23:PRINT STRING$(78,"●");
140 FOR I=3 TO 22:LOCATE 0,I:PRINT "●";
150 LOCATE 79,I:PRINT "●";:NEXT
160 LOCATE 0,2:PRINT "●";:LOCATE 79,2:PRINT "●";
170 LOCATE 0,23:PRINT "●";:LOCATE 79,23:PRINT "●";
180 F=1:X=RND(1)*77+1:C=RND(1)*18+3
190 FOR X=X TO 78
200  LOCATE XO,CO:PRINT " ";
210  IF CSRLIN=22 OR CSRLIN=3 THEN F=F*-1
220  C=C+F:LOCATE X,C:PRINT "●";:XO=X:CO=C
230 NEXT X
240 FOR X=78 TO 2 STEP-1
250  LOCATE XO,CO:PRINT " ";
260  IF CSRLIN=22 OR CSRLIN=3 THEN F=F*-1
270  C=C+F:LOCATE X,C:PRINT "●";:XO=X:CO=C
280 NEXT X
290 GOTO 190
```

# CVI／CVS／CVD

**機　能**　文字列を数値データに変換した値を与えます.

**書　式**　1）CVI(<2 文字の文字列>)

2）CVS(<4 文字の文字列>)

3）CVD(<8 文字の文字列>)

**文　例**　1）A%=CVI(A$)

2）B=CVS("A3Bd")

3）C#=CVD(NUM$)

**解　説**　ディスク上のランダムデータファイルから読み込んだデータは，文字型になっているため，CVI，CVS，CVD 各関数を使って，数値データに変換しなければなりません. CVI は 2 文字の文字列(即ち16ビットのデータ)を整数値に，CVS は 4 文字の文字列を単精度実数値に，CVD は 8 文字の文字列を倍精度実数値にそれぞれ変換します.

実際の文字列と変換後の数値との関係は，わかりやすく整数型で説明すると次の様になります.

A$＝"AB" の場合，即ち，A$＝CHR$(&H41)＋CHR$(&H42) の場合には，CVI(A$) は，&H4241(＝16961) になります. つまり，キャラクタコードを数値データとみなすわけです. CVS，CVD の場合は，各キャラクタコードを浮動小数点形式の数値データとみなすことになります.

参照：MKI$，MKS$，MKD$

**サンプルプログラム**

```
100 OPEN "RDATA" AS #1
110 FIELD #1,8 AS A$
120 PRINT "このプログラムで読み込むデータは
130 PRINT "ＭＫＩ＄／ＭＫＳ＄／ＭＫＤ＄　の
140 PRINT "サンプルプログラムで作られます
150 GET #1,1:A%=CVI(A$):PRINT A%
160 GET #1,2:A=CVS(A$):PRINT A
170 GET #1,3:A#=CVD(A$):PRINT A#
180 CLOSE:END


run
このプログラムで読み込むデータは
ＭＫＩ＄／ＭＫＳ＄／ＭＫＤ＄　の
サンプルプログラムで作られます
 987
 123.456
 1234567891234
Ok
```

# DATE$

機能　日付を与えます.

書式　DATE$

文例　D$＝DATE$

解説　DATE$には常に現在の日付が入れられており，いつでもその内容を見ることができます.

また DATE$ には，YY／MM／DD（"84／12／10" 等）の形で日付をセットできます．他の変数に代入した場合も，同じ形の文字列が代入されます.

ただし年に関しては DATE$ で更新しないと変化しません.

**注意**：PC-9800 シリーズパーソナルコンピュータはバッテリーバックアップにより常に日付を更新して正しい日付を維持するようになっています．従って不用意に日付を変えると，せっかくの正しい日付がこわされることになります．2 月は 28 日単位で更新されますので，うるう年については 2 月 29 日以降 DATE$ による日付の再設定が必要です.

**参照**：TIME$

サンプルプログラム

```
100 ' "85/12/21" --> "21,Dec.1985"
110  DIM MONTH$(12)
120  FOR I=1 TO 12
130   READ MONTH$(I)
140  NEXT I
150  DATE$="85/12/21"
160  GOSUB *TRNS
170  PRINT DATE$;" --> ";M$
180 END
190 *TRNS
200  M$=RIGHT$(DATE$,2)+","
210  M$=M$+MONTH$(VAL(MID$(DATE$,4,2)))
220  M$=M$+".19"+LEFT$(DATE$,2)
230  RETURN
240  DATA Jan,Feb,Mar,Apr,May,Jun
250  DATA Jul,Aug,Sep,Oct,Nov,Dec

run
85/12/21 --> 12,Dec.1985
Ok
```

# DSKF　　(DISK モード)

| | |
|---|---|
| 機　能 | ディスクの諸元を与えます. |
| 書　式 | DSKF(<ドライブ番号>〔, <機能>〕) |
| 文　例 | PRINT DSKF(1) |

解　説

<ドライブ番号>で指定されたディスクに関する情報を関数の値として返します. <機能>は値として返す情報の種類を指定します. <機能>として指定する値と, その時に返される関数の値の意味は次のようになっています.

省略された時：ディスクの残り容量をクラスタ単位で返す.

0：最大トラック番号（＝片面当りのトラック数－1）
1：1トラック当りのセクタ数
2：ディスクのサーフェイス(面)数－1
3：1トラック当りのクラスタ数
4：ボリューム当りのクラスタ数
5：ディレクトリトラック番号
6：1クラスタ当りのセクタ数
7：FAT の開始セクタ番号
8：FAT の終了セクタ番号
9：FAT の数
10：ディスク属性の入っているセクタ番号

ドライブの種類やディレクトリ, FAT 等については, ユーザーズマニュアルを参照してください.

# DSKI$　(DISK モード)

**機　能**　ディスクからの直接読み出しを行います.

**書　式**　DSKI$ (〈ドライブ番号〉, 〈ヘッド番号〉, 〈トラック番号〉, 〈セクタ番号〉)

**文　例**　D$=DSKI$ (2, 1, 19, 1)

**解　説**

通常のファイル操作(OPEN, CLOSE など)とは無関係に, ディスク上の指定したセクタから直接読み出しを行います.

DSKI$は, そのパラメータにより指定されたセクタ上に書かれているデータ256バイトをシステム用バッファに読み出すとともに, 最初の256文字の文字列を関数の値として返します.

文字列の長さは255文字までしか許されないために, 関数の値として256バイトのデータ全てを得ることはできません. そこで, 通常のランダムアクセスの場合と同じように FIELD 文により, システム用バッファへ複数の文字列変数を割り当てることによりこれを解決することができます. あるいは, VARPTR 関数により, システム用バッファの置かれているメモリ番地を調べた上で PEEK 関数によりバッファからデータを読み出すこともできます.

〈ヘッド番号〉, 〈トラック番号〉, 〈セクタ番号〉として指定できる値の範囲は, 指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが, BASIC は, これらの値の範囲を自動的に調べ, 不当であった場合には〝Bad track／sector″エラーとします. 詳しくはユーザーズマニュアルを参照してください.

参照：DSKO$, FIELD, DSKF, VARPTR

サンプルプログラム

```
100 '----- read from the disk (direct) ---
110  FIELD #0,128 AS A$(0), 128 AS A$(1)
120 *START
130  INPUT "Drive";DR
140  IF DR<1 OR DR>8 THEN *START
150  INPUT "Surface";SU
160  IF SU<0 OR SU>1 THEN 150
170 *ENTER
180  INPUT "Track,Secter";TR,SE
190  IF TR<0 OR TR>DSKF(DR,0) THEN *ENTER
200  IF SE<0 OR SE>DSKF(DR,1) THEN *ENTER
210  DUMMY$=DSKI$(DR,SU,TR,SE)
220 *DISP
230   FOR I=0 TO 1
240    FOR J=0 TO 7
250     FOR K=1 TO 16
260       D$=HEX$(ASC(MID$(A$(I),J*16+K,1)))
270       PRINT RIGHT$("0"+D$,2);" ";
280     NEXT K:PRINT
290   NEXT J,I:END
```

# EOF

機能　ファイルの終了コードを与えます．

書式　EOF(<ファイル番号>)

文例　IF EOF(1) THEN CLOSE 1 : END

解説　<ファイル番号>で指定されたファイルが終りに達したかどうかを調べる関数です．関数の値として終りに達していれば真(－1)，そうでなければ偽(0)を返します．<ファイル番号>で指定されたファイルは入力モードでオープンされていなければなりません．

指定されたファイルがRS-232Cコミュニケーションチャネル（COM：）であった場合には，EOF関数は，バッファが空であったときに値を真とします．

サンプルプログラム

```
100 '-- data input --
110 OPEN "2:DATA5" FOR INPUT AS #1
120 IF EOF(1) THEN 160
130  INPUT #1,A
140  PRINT A
150 GOTO 120
160 CLOSE

run
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
Ok
```

# ERL／ERR

機　能　発生したエラーのエラーコード及びエラーの発生した行番号を与えます．

書　式　ERL

ERR

文　例　IF ERL＝100 THEN STOP

IF ERR＝7 THEN STOP

解　説　エラーが発生した時点で，システム変数ERRはエラーコードを，ERLはエラーの発生した行番号を持っています．

一般に，ERR及びERLはON ERROR GOTO文によって指定したエラー処理ルーチンにおいて，処理の流れを制御するために使われます．

参照：ON ERROR GOTO

サンプルプログラム

```
100 '－－　 Xの Y 乗を求める　 －－
110  ON ERROR GOTO *ERRORTRP
120 *START
130  INPUT "X,Y";X,Y
140  ANS=X^Y
150 PRINT X;"の";Y;"乗は";ANS;"です．"
160  GOTO *START
170  PRINT "？？？　定義できません．"
180  END
190 '-- error routine --
200 *ERRORTRP
210  IF ERR=11 AND ERL=140 THEN RESUME 170
220  ON ERROR GOTO 0


run
X,Y? 2,3
 2 の 3 乗は 8 です．
X,Y? 10,4
 10 の 4 乗は 10000 です．
X,Y? -1,3
-1 の 3 乗は-1 です．
X,Y? 0,-1
？？？　定義できません．
Ok
```

# EXP

機能　eに対する指数関数の値を与えます．

書式　EXP(〈数式〉)

文例　E＝EXP(1)

解説　〈数式〉の指数関数の結果を値とします．

〈数式〉に倍精度実数が含まれる場合は倍精度の値を与えますが，他の場合には単精度の値を与えます．

サンプルプログラム

```
100 ' EXP から SINH(ハイパブリック  サイン)
110 ' COSH(ハイパブリック  コサイン)を求める
120 INPUT D
130 HS=(EXP(D)-EXP(-D))/2:HC=(EXP(D)+EXP(-D))/2
140 PRINT "SINH";USING "###";D;:PRINT " = ";HS,
150 PRINT "COSH";USING "###";D;:PRINT " = ";HC,
160 END

run
? 2
SINH  2 =  3.62686          COSH  2 =  3.7622
Ok
```

# FIX

| | |
|---|---|
| 機　能 | 整数部を与えます. |
| 書　式 | FIX(〈数式〉) |
| 文　例 | F＝FIX(B／3) |

解　説

〈数式〉の値の小数点以下を取り去った値を与えます.

FIX と INT の違いは, 〈数式〉の値が負の時に, INT は〈数式〉より小さい最大の整数を返すのに対し, FIX はたんに小数点部分だけをとり去った値を返すことです.

参照：INT

サンプルプログラム

```
100 '-- fix & int --
110 A=-2
120 FOR I=0 TO 9
130 A=A+I*.3
140 PRINT "A=";A,"FIX(A)=";FIX(A),"INT(A)=";INT(A)
150 NEXT I


run
A=-2            FIX(A)=-2       INT(A)=-2
A=-1.7          FIX(A)=-1       INT(A)=-2
A=-1.1          FIX(A)=-1       INT(A)=-2
A=-.2           FIX(A)= 0       INT(A)=-1
A= 1            FIX(A)= 1       INT(A)= 1
A= 2.5          FIX(A)= 2       INT(A)= 2
A= 4.3          FIX(A)= 4       INT(A)= 4
A= 6.4          FIX(A)= 6       INT(A)= 6
A= 8.8          FIX(A)= 8       INT(A)= 8
A= 11.5         FIX(A)= 11      INT(A)= 11
Ok
```

# FPOS

機　能　ファイル中での物理的な現在位置を与えます.

書　式　FPOS (<ファイル番号>)

文　例　HEAD=FPOS(4)

解　説　<ファイル番号>によって指定されたファイルが最後に読み書きした位置を返します.

指定されたファイルがディスクファイルの場合には，最後に読み書きしたセクタの番号を返します．この番号はヘッド０，トラック０，セクタ１を０番とした通し番号で与えられます.

指定されたファイルがプリンタであった場合には，プリンタのヘッドの位置を与えます．この値はLPOSが返す値と同一です.

参照：LOC，LPOS

# FRE

**機能** **メモリの未使用領域の大きさを与えます.**

**書式** FRE(<機能>)

**文例** PRINT FRE(0)

**解説** FREは,BASICの利用可能なメモリ領域のうちの,未使用領域の大きさを関数値として返します.〈機能〉は, 0, 1, 2, 3の値をとり,それぞれの場合に返される関数値の意味は次のとおりです.

0:未使用変数領域のバイト数.単純変数およびストリングの領域として使用可能な領域の残りバイト数です.

1:未使用テキスト領域のバイト数.プログラムテキストを入れるための領域の残りバイト数です.

2:未使用変数領域と未使用テキスト領域のバイト数.

3:配列データセグメントの未使用領域のバイト数.

〈機能〉が0または2の時は,FRE関数は,ストリング領域の整理のための〝ちり集め〟処理を必ず行いますので,関数が値を返すまでに時間がかかることがあります.

F

# HEX$

機　能　10進数を16進数に変換し，その文字列を与えます．

書　式　HEX$(〈数式〉)

文　例　A$＝HEX$(X)

解　説　〈数式〉の値を16進数に変換して，その文字列を与える関数です．〈数式〉の値は，－32768から32767（または0から65535）までの範囲になければなりません．

参照：VAL，OCT$

サンプルプログラム

```
100 PRINT "     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9"
110 FOR I=0 TO 9
120 PRINT STR$(I);"- ";
130  FOR J=0 TO 9
140   PRINT RIGHT$("0"+HEX$(I*10+J),2);" ";
150  NEXT J
160  PRINT
170 NEXT I
180 END
```

```
run
     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9
 0-  00 01 02 03 04 05 06 07 08 09
 1-  0A 0B 0C 0D 0E 0F 10 11 12 13
 2-  14 15 16 17 18 19 1A 1B 1C 1D
 3-  1E 1F 20 21 22 23 24 25 26 27
 4-  28 29 2A 2B 2C 2D 2E 2F 30 31
 5-  32 33 34 35 36 37 38 39 3A 3B
 6-  3C 3D 3E 3F 40 41 42 43 44 45
 7-  46 47 48 49 4A 4B 4C 4D 4E 4F
 8-  50 51 52 53 54 55 56 57 58 59
 9-  5A 5B 5C 5D 5E 5F 60 61 62 63
Ok
```

# INKEY$

| 機 能 | キーが押されていれば，その文字を，キーが押されなければヌルストリングを与えます． |
|---|---|
| 書 式 | INKEY$ |
| 文 例 | A$＝INKEY$ |

解 説

キーボードバッファが空であれば，INKEY$関数はヌルストリング(空の文字列)を返します．キー入力によりキーボードバッファが空でない場合には，バッファの先頭から1文字取り出し，その文字を返します．

なお，CTRL-C，STOP キーは INKEY$関数によって拾うことはできません．また押されたキーは画面上には表示されません．

サンプルプログラム

```
100  PRINT "Enter any key to end ･･･"
110 *LOOP
120  A$=INKEY$
130  IF A$="" THEN *LOOP
140  PRINT A$
150  IF A$="E" OR A$="e" THEN *EXIT ELSE *LOOP
160 *EXIT:END
```

```
run
Enter any key to end ･･･
n
8
8
-
b
a
s
i
c

e
```

# INP

| | |
|---|---|
| 機　能 | 入力ポートから値を得ます. |
| 書　式 | INP(<ポート番号>) |
| 文　例 | A=INP(15) |

解　説

<ポート番号>で指定された入力ポートから8ビットのデータを読み取り，それを関数値とします.<ポート番号>として指定できる値の範囲は0から32767までです. <ポート番号>と装置との対応は各本体に添付されているユーザーズマニュアルを参照してください.

参照：OUT

サンプルプログラム

```
100 PRINT "Type 'e' key to end ···"
110 *LOOP
120  IF INP(&HE2)=223 THEN END
130 '  We have to dummy read
140 ' to clear the key buffer.
150 DUM$=INPUT$(1):PRINT DUM$
160 GOTO *LOOP

run
Type 'e' key to end ···
a
b
c
d
e
Ok
```

# INPUT$

| | |
|---|---|
| 機 能 | **指定されたファイルより指定された長さの文字列を入力し，与えます．** |
| 書 式 | INPUT$(<文字数>〔,〔#〕<ファイル番号>〕) |
| 文 例 | WORD$=INPUT$(6, #2) |

解 説

<ファイル番号>によって指定されたファイルから<文字数>分の文字列を読み出します．<ファイル番号>が省略された場合には，キーボードから入力が行われますが，INPUT 文と異なり入力された文字はスクリーンには表示されません．

INPUT$は指定された<文字数>の文字が入力されるのを待ち続けますが，既にバッファに入力済みのデータがある場合にはバッファ中から文字を拾ってきます．またINPUT$は，STOP キー，CTRL-C を除くすべての文字をそのまま読み出しますので，INPUT 文や LINE INPUT 文では入力することのできない改行文字等も入力することができます．

参照：INPUT，LINE INPUT

# INSTR

**機　能**　**文字列の中から任意の文字列を捜してその文字の位置を与えます．**

**書　式**　INSTR(〔〈数式〉,〕〈文字列1〉,〈文字列2〉)

**文　例**　J=INSTR(A$, "J")

**解　説**　〈文字列1〉の中から〈文字列2〉を探し，みつかれば発見した位置を，見つからなければ0を値として返します．

〈数式〉は捜し始める位置を指定します．もし省略した場合は〈文字列1〉の最初から捜し始めます．

〈文字列2〉に "" (空の文字列)を指定すると，〈数式〉と同じ値を返します．

**サンプルプログラム**

```
100 '-- alphabet search --
110  A$="abcdefghijklmnopqrstuvwxyz"
120  ALP$=INPUT$(1)
130  ALP=ASC(ALP$)
140   IF ALP>64 AND ALP<91 THEN ALP=ALP+32
150  A=INSTR(A$,CHR$(ALP))
160   IF A=0 THEN PRINT ALP$;" は  アルファベット  ではありません":END
170  PRINT ALP$;" は ";A;" 番目のアルファベットです"
180 END

run
S は  19  番目のアルファベットです
Ok
```

# INT

機 能　小数点以下を切り捨てた整数値を与えます.

書 式　INT(〈数式〉)

文 例　PRINT INT(1.123)

解 説　〈数式〉の値を超えない最大の整数を与えます.

参照：FIX, CINT

サンプルプログラム

```
100 '  整数と実数の区別
110  A$(0)="実数です"
120  A$(1)="整数です"
130  FLG=0
140  INPUT A
150   IF A=INT(A) THEN FLG=1
160  PRINT A;"  は  ";A$(FLG)
170 END


run
? 123
 123   は  整数です
Ok
run
? 88.995
 88.995   は  実数です
Ok
```

# JIS$　（DISK モード）

機　能　**日本語文字を漢字コードで与えます.**

書　式　JIS$(<文字列>)

文　例　A$=JIS$(B$)

解　説　〈文字列〉の最初の2バイトを16進表記4桁の文字列に変換して与えます.

この関数は，日本語文字を漢字コードで与えるためのものですが，最初の1バイトが日本語コードとして妥当な値の場合，後続の1バイトを含めた最初の2バイトを16進表記4桁に変換し，妥当でない場合最初の2バイトを16進4桁の文字列に変換します.

〈文字列〉の内容が2バイト以下の場合は，"Illegal function call"エラーになります.

サンプルプログラム

```
100 PRINT JIS$(KMID$("漢字",2,1))

run
3441
Ok
```

# KACNV$　（DISK モード）

機能　**文字列の中の日本語文字を対応する英数カナ(1バイト系)文字に変換します.**

書式　KACNV$(<文字列>)

文例　A$＝KACNV$(B$)

解説　〈文字列〉の中の1バイト系英数カナ文字は変換せず，2バイト系の日本語文字を1バイト系英数カナ文字に変換します.

なお，日本語文字列中の KI／KO コードはすべて取りのぞきます.

サンプルプログラム

```
100 A$="イロハＡＢＣ"
110 PRINT KACNV$(A$)

run
ｲﾛﾊABC
Ok
```

J

K

# KEXT$　(DISK モード)

機　能　文字列の中から１バイト系英数カナ文字だけ，あるいは，２バイト系日本語文字だけのどちらかを抜き出して与えます．

書　式　KEXT$(〈文字列〉，〈機能〉)

文　例　A$＝KEXT$(B$，１)

解　説　〈機能〉指定が〝０″なら〈文字列〉の中の１バイト系英数カナ文字だけ抜き出し，〝１″なら２バイト系日本語文字を抜き出します．この時，KI／KO コードは抽出されません．

〈機能〉で指定したタイプの文字が無い場合には〝″(空の文字列)を与えます．

サンプルプログラム

```
100 A$="ABC漢字DEF"
110 KI$=CHR$(&H1B)+CHR$(&H4B):KO$=CHR$(&H1B)+CHR$(&H48)
120 PRINT KI$+KEXT$(A$,1)+KO$

run
漢字
Ok
```

# KINSTR　（DISK モード）

**機　能**　日本語文字を含む文字列の中から指定した文字列を捜し，その文字の位置を与えます．

**書　式**　KINSTR(〔〈数式〉,〕〈文字列1〉, 〈文字列2〉)

**文　例**　J=KINSTR(3, A$, "東京")

**解　説**　〈文字列1〉の中から〈文字列2〉を探し出し，見つかれば発見した位置を，見つからなければ0を値として返します．

〈数式〉は捜し始める位置を指定するもので(捜し始める位置は，文字数(日本語も1文字として数える)で表し，KI／KOコードを文字数に含めます)，省略した場合は，〈文字列1〉の最初から探し，〈文字列1〉の文字数より大きな値なら0を値として返します．

〈文字列2〉が "" (空の文字列)の場合は〈数式〉と同じ値を返します．

**サンプルプログラム**

```
100 KINPUT A$
110 B=KINSTR(A$,"港区")
120 IF B<>0 THEN PRINT "港区" ELSE PRINT "港区以外"
130 GOTO 100

run
東京都北区赤羽8丁目
港区以外
東京都港区南青山6丁目
港区

Break in 100
Ok
```

# KLEN　（DISKモード）

**機　能**　**日本語を含む文字列の文字数を与えます.**

**書　式**　KLEN(〈文字列〉〔, 〈機能〉〕)

**文　例**　N=KLEN(A$, 0)

**解　説**　〈機能〉で指定したタイプの文字の数を与えるもので，〈機能〉としては次のものを指定します.

0……全体の文字数（日本語文字は2バイトで1文字．KI／KOコードを含みます）

1……1バイト系英数カナの文字数

2……2バイト系日本語の文字数（KI／KOコードは含みません）

3……2バイト系全角文字の文字数

4……2バイト系半角文字の文字数

5……KI／KOコードの数

〈機能〉を省略した場合は0と解釈されます.

**サンプルプログラム**

```
100 A$="ABC漢字DE日本語"
110 PRINT KLEN(A$,0),KLEN(A$,1),KLEN(A$,2),KLEN(A$,5)

run
 14            5             5             4
Ok
```

# KMID$　（DISK モード）

**機　能**　**日本語を含む文字列の中から任意の長さの文字列を与えます．**

**書　式**　KMID$(〈文字列〉, 〈式1〉〔, 〈式2〉〕)

**文　例**　K$=KMID$(A$, 4, 3)

**解　説**　〈文字列〉の〈式１〉番目の文字から始まる〈式２〉文字を与えます．ここでの文字数は，１バイト系英数カナ文字は１バイトを１文字とし，２バイト系日本語文字およびKI／KOコードは，２バイトを１文字として数えます．

〈文字列〉の文字数が〈式１〉より小さい場合は，“”（空の文字列）を与えます．

〈式２〉を省略した場合や，〈文字列〉の〈式１〉番目の文字から右の文字数が〈式２〉より小さい場合は，〈文字列〉の〈式１〉番目より右のすべての文字列を結果として与えます．

**サンプルプログラム**

```
100 A$="ABC漢字DEF"
110 B$="":C$=""
120 FOR I=1 TO KLEN(A$,0)
130 T=KTYPE(A$,I)
140    IF T=0 THEN B$=B$+KMID$(A$,I,1)
150    IF T=1 OR T=2 THEN C$=C$+KMID$(A$,I,1)
160 NEXT
170 PRINT B$,CHR$(&H1B)+CHR$(&H4B)+C$+CHR$(&H1B)+CHR$(&H48)

run
ABCDEF        漢字
Ok
```

# KNJ$ (DISK モード)

機能　漢字コード文字列4桁を日本語1文字に変換して与えます.

書式　KNJ$(<文字列>)

文例　KNJ$("4E7A")

解説　〈文字列〉の最初の4桁を日本語1文字に変換して与えます. この時, 文字列の内容が次の条件に当てはまらない場合には, "Illegal function call" エラーになります.

(1) 文字列が4桁以上
(2) 4桁の文字に0～9, A～Fの範囲の文字
(3) 4桁の文字の組み合わせが, 漢字コードの範囲あるいはKI／KOコード

サンプルプログラム

```
100 PRINT KNJ$("1B4B")+KNJ$("3441")+KNJ$("1B48")

run
漢
Ok
```

# KTYPE　　（DISK モード）

**機　能**　**日本語を含む文字列の中の指定位置の文字のタイプを与えます.**

**書　式**　KTYPE(〈**文字列**〉, 〈**式**〉)

**文　例**　T=KTYPE(A$, 5)

**解　説**

〈文字列〉の〈式〉番目の文字のタイプを次の順で与えます.

0……1バイト系英数カナ文字

1……2バイト系全角日本語文字

2……2バイト系半角文字

3……KIコード

4……KOコード

〈式〉の値が0または文字列の文字数より大きな場合は, "Illegal function call" エラーになります. 〈文字列〉が"" (空の文字列)の場合も, "Illegal function call" エラーになります.

**サンプルプログラム**

```
100 A$="ABC漢字DE"
110 FOR I=1 TO KLEN(A$,0)
120 PRINT KTYPE(A$,I);
130 NEXT

run
 0  0  0  3  1  1  4  0  0
Ok
```

# LEFT$

機能　文字列の左側から任意の長さの文字列を与えます．

書式　LEFT$(〈文字列〉，〈数式〉)

文例　B$=LEFT$(A$, 4)

解説　〈数式〉は 0 から255の範囲になければなりません．〈数式〉が，〈文字列〉の総文字数以上の場合は，〈文字列〉のすべてを結果とします．〈数式〉が 0 ならば，空の文字列を結果とします．

サンプルプログラム

```
100 '  文字列（10文字）の左回転
110  INPUT A$
120  FOR I=1 TO 10
130   PRINT A$
140   B$=LEFT$(A$,1)
150   A$=MID$(A$,2)+B$
160  NEXT I
170 END
```

```
run
? 1234567890
1234567890
2345678901
3456789012
4567890123
5678901234
6789012345
7890123456
8901234567
9012345678
0123456789
Ok
```

# LEN

機能　文字列の文字数を与えます.

書式　LEN(<文字列>)

文例　PRINT LEN(A$)

解説　出力されない文字や空白も数えます．出力されない文字とはキャラクタコードで0から31までの，通常コントロールコードと呼ばれるものをいいます．

L

# LOC

**機　能**　ファイル中での論理的な現在位置を与えます．

**書　式**　LOC(〈ファイル番号〉)

**文　例**　LAST＝LOC(2)

**解　説**　〈ファイル番号〉で指定されたファイルがランダムディスクファイルであった場合には，LOC 関数は，そのランダムファイルに対して最後に読み書き(GET/PUT)されたレコードのレコード番号を返します．

〈ファイル番号〉で指定されたファイルが，シーケンシャルディスクファイルであった場合には，そのファイルがオープンされてから，読み書きされたレコード数を返します．

指定されたファイルが RS-232C コミュニケーションファイルであった場合には，LOC 関数は，割り込みによって受けつけられ，入力バッファ中にたまっている文字数を値として返します．読み取り動作(INPUT #，INPUT$等)が行われずに，割り込みによる RS-232C チャネルからの受信が続くと，やがてバッファが一杯になりエラーとなります．

指定されたファイルがキーボードファイルであった場合には，LOC 関数はキーボードからの割り込みによって受けつけられて，キーボードバッファにたまっている文字数を返します．

**サンプルプログラム**

```
100 OPEN "DATA6" AS #1
110 FIELD #1,20 AS A$,8 AS B$
120 INPUT "名前";NA$:INPUT "給与";KY#
130 LSET A$=NA$:RSET B$=MKD$(KY#):PUT #1
140 PRINT NA$;"さんのデータは";AKCNV$(STR$(LOC(1)));"番です"
150 INPUT "Continue?(Y/N)";C$
160 IF C$="y" OR C$="Y" THEN PRINT:GOTO 120
170 IF C$="n" OR C$="N" THEN END ELSE 150


run
名前? 益田　文
給与? 155000
益田　文さんのデータは 1 番です
Continue?(Y/N) N
Ok
```

# LOF

| | |
|---|---|
| 機 能 | **ファイルの大きさを与えます.** |
| 書 式 | LOF(<ファイル番号>) |
| 文 例 | MAXREC=LOF(2) |

解 説　<ファイル番号>により指定されたファイルがディスクファイルである場合には，LOF関数はそのファイルの大きさをセクタ数で返します．この値は，ファイルがラムダムファイルであれば，そのファイルの最大レコード番号に相当します．

指定されたファイルがRS-232Cコミュニケーション・ファイルである場合には，LOF関数は，バッファの残りバイト数を返します．

L

# LOG

機 能　自然対数を与えます.

書 式　LOG(<数式>)

文 例　PRINT LOG(35／9)

解 説　<数式>によって与えられた値の自然対数(e を底とした対数)を返します.

また, <数式>に倍精度実数が含まれる場合は, 倍精度の値を与えますが, 他の場合には, 単精度の値を与えます.

参照：EXP

サンプルプログラム

```
100 '  常用対数を求める
110  A=LOG(10)
120  F$="Log10(##.#####^^^^)=##.#####^^^^"
130  INPUT L
140  A=LOG(10)
150  B=LOG(L)
160  PRINT USING F$;L,B/A
170 END


run
? 100
Log10( 1.00000E+02)= 2.00000E+00
Ok
run
? .1245
Log10( 1.24500E-01)=-9.04831E-01
Ok
```

# LPOS

**機　能**　現在のプリンタのヘッド位置を与えます.

**書　式**　LPOS(<式>)

**文　例**　LPOS(0)

**解　説**　<式>の値はダミーであり意味を持ちません(ただし，省略できません).

参照：WIDTH LPRINT

**サンプルプログラム**

```
100 WIDTH "LPT1:",255
110 FOR I=0 TO 20
120  IF LPOS(0)=>40 THEN LPRINT
130  LPRINT SQR(I),
140 NEXT I
150 END
```

```
 0              1              1.41421
 1.73205        2              2.23607
 2.44949        2.64575        2.82843
 3              3.16228        3.31662
 3.4641         3.60555        3.74166
 3.87298        4              4.12311
 4.24264        4.3589         4.47214
```

# MAP

| | |
|---|---|
| 機　能 | **スクリーン座標，ワールド座標の相互変換を行います．** |
| 書　式 | MAP(〈数式〉，〈機能〉) |
| 文　例 | SY1＝MAP(WY1，1) |

解　説

〈数式〉で指定されるスクリーン座標あるいはワールド座標を，〈機能〉の指定に従い，対応するワールド座標あるいはスクリーン座標に変換します．

〈機能〉は変換の種類を指定し，0から3までの値をとり，それぞれ次の意味を持ちます．

0：〈数式〉をワールド座標系における x 座標とみなし，これを対応するスクリーン座標系での x 座標に変換します．(Wx → Sx)

1：〈数式〉をワールド座標系における y 座標とみなし，これを対応するスクリーン座標系での y 座標に変換します．(Wy → Sy)

2：〈数式〉をスクリーン座標系における x 座標とみなし，これを対応するワールド座標系での x 座標に変換します．(Sx → Wx)

3：〈数式〉をスクリーン座標系における y 座標とみなし，これを対応するワールド座標系での y 座標に変換します．(Sy → Wy)

MAP 関数を用いれば，ワールド座標からスクリーン座標の変換ができますから，GET 文や PUT 文のように，スクリーン座標で位置を指定する操作の場合にもワールド座標を変換して指定することができます．

**注意**：MAP 関数は，変換の結果がスクリーン座標系あるいはワールド座標系から外れてもエラーにならないので注意してください．

サンプルプログラム

```
100 '** World <--> Screen **
110  SCREEN 0,0
120  WINDOW(-100,-100)-(100,100)
130  PRINT TAB(5);"Window(-100,-100)-(100,100)"
140  PRINT TAB(5);"View(0,0)-(639,199)"
150  PRINT "においてＷＯＲＬＤ＜ーー＞ＳＣＲＥＥＮの変換を行います．"
160  PRINT
170 *ENTRY
180  PRINT "World --> Screen : 1"
190  PRINT "Screen --> World : 2"
200  INPUT ANS
210  IF ANS<>1 AND ANS<>2 THEN *ENTRY
220  INPUT "Coordinate =";CO
230  IF ANS=2 THEN *WFROMS
240  PRINT "WX :";CO,"SX =";MAP(CO,0)
250  PRINT "WY :";CO,"SY =";MAP(CO,1)
260  GOTO *ENTRY
270 *WFORMS
280  PRINT "SX :";CO,"WX =";MAP(CO,2)
290  PRINT "SY :";CO,"WY =";MAP(CO,3)
300  GOTO *ENTRY
```

```
run
     Window(-100,-100)-(100,100)
     View(0,0)-(639,199)
においてＷＯＲＬＤ＜ーー＞ＳＣＲＥＥＮの変換を行います．

World --> Screen : 1
Screen --> World : 2
? 1
Coordinate =? 0
WX : 0          SX = 320
WY : 0          SY = 100
World --> Screen : 1
Screen --> World : 2
? 2
Coordinate =? 100
SX : 100        WX =-68.7011
SY : 100        WY = .50251
World --> Screen : 1
Screen --> World : 2
?
Break in 200
Ok
```

# MID$

機能　**文字列の中から任意の長さの文字列を与えます．**

書式　MID$(〈文字列〉, 〈式1〉〔, 〈式2〉〕)

文例　PRINT MID$(A$, 2, 3)

解説　〈文字列〉の〈式1〉番目の文字から〈式2〉桁の文字列を与えます．

〈式2〉は0から255の範囲，また，〈式1〉は1から255の範囲になければなりません．

また，〈式2〉を省略した場合や，〈文字列〉の〈式1〉番目の文字から右の文字数が〈式2〉より小さい場合は，〈文字列〉の〈式1〉番目より右すべての文字列を結果とします．

〈文字列〉の文字数が，〈式1〉より小さければ，MID$関数は，空の文字列を与えます．

参照：RIGHT$，LEFT$

サンプルプログラム

```
100 '-- break of string --
110  INPUT A$
120  PRINT A$;" = ";
130  FOR I=1 TO LEN(A$) STEP 2
140   PRINT MID$(A$,I,2);" + ";
150  NEXT
160  PRINT CHR$(8);CHR$(8);" "
170 END

run
? 1234567890ABCDE
1234567890ABCDE = 12 + 34 + 56 + 78 + 90 + AB + CD + E
Ok
```

# MKI$／MKS$／MKD$

機　能　各数値を内部表現に対応した文字コードに変換します.

書　式　1）MKI$(<整数表記>)

2）MKS$(<単精度表記>)

3）MKD$(<倍精度表記>)

文　例　1）A$＝MKI$(128)

2）LSET B$＝MKS$(1.23)

3）RSET C$＝MKD$(3.141592654＃)

解　説　これらの関数は，数値をランダムファイルバッファに対して LSET／RSET 命令で書き込む際に使用します.数値から文字への変換は数値が持つ内部表現(2進数表現）の値をそのままそれに対応する文字コードにすることによって行われます.この逆の動作をする関数としては CVI／CVS／CVD が用意されています.

書式1）整数値を2文字(2バイト)の文字列に変換します.

書式2）単精度数値を4文字(4バイト)の文字列に変換します.

書式3）倍精度数値を8文字(8バイト)の文字列に変換します.

注意：STR$関数とは変換の仕方が異なります.

参照：CVI，CVS，CVD，STR$

サンプルプログラム

```
100 OPEN "RDATA" AS #1
110 FIELD #1,8 AS A$
120 A%=987:LSET A$=MKI$(A%):PUT #1
130 A!=123.456:LSET A$=MKS$(A!):PUT #1
140 A#=1234567891234#:LSET A$=MKD$(A#):PUT #1
150 PRINT "このプログラムで書き込むデータは
160 PRINT "ＣＶＩ／ＣＶＳ／ＣＶＤ　の
170 PRINT "サンプルプログラムで読み出せます
180 CLOSE:END

run
このプログラムで書き込むデータは
ＣＶＩ／ＣＶＳ／ＣＶＤ　の
サンプルプログラムで読み出せます
Ok
```

M

# OCT$

**機 能**　10進数を８進数に変換し，その文字列を与えます．

**書 式**　OCT$(<数式>)

**文 例**　PRINT OCT$(A)

**解 説**　<数式>の値の範囲は−32768から32767(または０から65535)までです．

参照：HEX$

**サンプルプログラム**

```
100 '   １０進ーー＞８進ーー＞１６進変換
110 FOR I=0 TO 16
120   DE$=RIGHT$("   "+STR$(I),3)
130   OC$=RIGHT$("   "+OCT$(I),3)
140   HE$=RIGHT$("   "+HEX$(I),3)
150  PRINT "DECIMAL:";DE$,"OCTAL:";OC$,"HEX:";HE$
160 NEXT I


run
DECIMAL:  0   OCTAL:  0    HEX:  0
DECIMAL:  1   OCTAL:  1    HEX:  1
DECIMAL:  2   OCTAL:  2    HEX:  2
DECIMAL:  3   OCTAL:  3    HEX:  3
DECIMAL:  4   OCTAL:  4    HEX:  4
DECIMAL:  5   OCTAL:  5    HEX:  5
DECIMAL:  6   OCTAL:  6    HEX:  6
DECIMAL:  7   OCTAL:  7    HEX:  7
DECIMAL:  8   OCTAL: 10    HEX:  8
DECIMAL:  9   OCTAL: 11    HEX:  9
DECIMAL: 10   OCTAL: 12    HEX:  A
DECIMAL: 11   OCTAL: 13    HEX:  B
DECIMAL: 12   OCTAL: 14    HEX:  C
DECIMAL: 13   OCTAL: 15    HEX:  D
DECIMAL: 14   OCTAL: 16    HEX:  E
DECIMAL: 15   OCTAL: 17    HEX:  F
DECIMAL: 16   OCTAL: 20    HEX: 10
Ok
```

# PEEK



機　能　メモリ上の指定された番地の内容を読み出します.

書　式　PEEK(<番地>)

文　例　A＝PEEK(&H100)

解　説　<番地>には DEF SEG 文で指定したベースアドレスからの相対アドレスを指定します.

参照：CLEAR, DEF SEG, POKE, VARPTR

# PEN

**機能** ライトペンからの情報を与えます.

**書式** PEN(<機能>)

**文例** X=PEN(1)

**解説**

ライトペンが，現在どのような状態にあるかの情報を与えます.

ライトペンからの情報には，ライトペンが押され，光を感じた瞬間のもの(トリガセンス)とライトペンが押されて光を感じている間のもの(レベルセンス)があり，<機能>の値によって次のように異なります.

PEN(0)：トリガセンス

ペンが押され，光を感じたら真(－1)を，ペンが押され，光を感じないと偽(0)を与えます．この関数は，一度真になるとON PEN GOSUB文で定義されている割り込み処理ルーチンに入るまでその値を保持し，処理ルーチンに入った時，偽となります.

PEN(1)：レベルセンス

ペンが押され光を感じている限り，現在ペンが動作しているキャラクタ座標の水平座標を与えます.

PEN(2)：レベルセンス

ペンが押され光を感じている限り，現在ペンが動作しているキャラクタ座標の垂直座標を与えます.

**注意**：<式>の値はここで示す0～2までで，それ以外はエラーとなります．またこの関数はPEN ONの状態でなければ値を返しません.

**参照**：ON PEN GOSUB，PEN ON／OFF／STOP

サンプルプログラム

```
100 SCREEN 0,1:CONSOLE ,,,1
110 COLOR 0,7:CLS 2
120 PEN ON:ON PEN GOSUB *PENSUB
130 GOTO 130
140 *PENSUB:PRINT "X:";PEN(1),"Y:";PEN(2)
150 RETURN
```

```
run
X: 26          Y: 3
X: 55          Y: 14
Ok
```

# 〔1〕POINT



| | |
|---|---|
| 機能 | Last Referenced Point (LP)の値を与えます. |
| 書式 | POINT(<機能>) |
| 文例 | LX=POINT(0) |

**解説**

最後にグラフィック操作の行われた座標(LP)を<機能>の指定により，スクリーン座標あるいは，ワールド座標で与えます.

<機能>は0から3の整数値をとり，それぞれの場合にPOINT関数の返す値の意味は次のとおりです.

0：最後にグラフィック操作の行われた点のX座標をワールド座標系上の値に変換して返す.

1：同じくY座標を返す.

2：最後にグラフィック操作の行われた点のX座標を，スクリーン座標系上の値に変換して返す.

3：同じくY座標を返す.

最後に操作の行われた座標がウィンドウの外であった場合には，POINT関数の返す座標に対応する点は，画面上に表示されていない点であることに注意してください．またこの時，スクリーン座標系に変換されて返された値である場合には，ビューポート外になることに注意してください.

参照：POINT文

**サンプルプログラム**

```
100 SCREEN 0,0:CLS 3:GOSUB *PRINTLP
110 PSET(100,100):GOSUB *PRINTLP
120 WINDOW(-100,-100)-(100,100)
130 PSET(50,50):GOSUB *PRINTLP:END
140 *PRINTLP:PRINT:PRINT "World  coordinate: ";
150  PRINT "X =";POINT(0),"Y =";POINT(1)
160  PRINT "Screen coordinate: ";
170  PRINT "X =";POINT(2),"Y =";POINT(3)
180 RETURN
```

```
World  coordinate: X = 0    Y = 0
Screen coordinate: X = 0    Y = 0

World  coordinate: X = 100  Y = 100
Screen coordinate: X = 100  Y = 100

World  coordinate: X = 50   Y = 50
Screen coordinate: X = 479  Y = 149
Ok
```

## 〔2〕 POINT

**機 能**　**スクリーン座標上の指定された座標に表示されているドットの色を与えます.**

**書 式**　POINT(Sx, Sy)

**文 例**　C=POINT(473, 120)

**解 説**　(Sx, Sy)により指定されたスクリーン座標上に表示されているドットの色をパレット番号で返します. また, 指定された座標(Sx, Sy)がビューポートの外にある場合には, POINT 関数は－1を返します.

座標はスクリーン座標で指定することに注意してください. 白黒モードの場合には, POINT 関数は, リセット(黒)またはセット(白)をドットの色として返します.

**サンプルプログラム**

```
100 '-- change color painting --
110 DEF FNX=RND*639:DEF FNY=RND*199
120 DEF FNC=INT(RND*7+1)
130 SCREEN 0,0:CLS 3
140 '-- circle paint --
150 FOR I=1 TO 20
160  X=FNX:Y=FNY:R=RND*80+20:C=FNC
170  CIRCLE(X,Y),R,7:PAINT(X,Y),C,7
180 NEXT I
190 '-- change color --
200 FOR I=1 TO 50
210  X=FNX:Y=FNY:C=FNC:CC=POINT(X,Y)
220  IF CC=0 OR CC=C THEN 240
230  PAINT(X,Y),C,7
240 NEXT I
```

# POS

**機能** テキスト画面上の現在のカーソルの水平位置を与えます.

**書式** POS(〈式〉)

**文例** P=POS(0)

**解説** 式はダミーです(省略することはできません).
与えられる値はキャラクタ座標の水平位置です.

参照:CSRLIN

**サンプルプログラム**

```
100 FOR I=32 TO 255
110  IF POS(0)=>32 THEN PRINT
120  PRINT CHR$(I);" ";
130 NEXT I


run
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ ¥ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { ¦ } ~
▁ ▂ ▃ ▄ ▅ ▆ ▇ █ ▏ ▎ ▍ ▌ ▋ ▊ ▉ ┼
┴ ┬ ┤ ├ ▔ ─ │ ▕ ┌ ┐ └ ┘ ╭ ╮ ╰ ╯
  。 「 」 、 ・ ヲ ァ ィ ゥ ェ ォ ャ ュ ョ ッ
ー ア イ ウ エ オ カ キ ク ケ コ サ シ ス セ ソ
タ チ ツ テ ト ナ ニ ヌ ネ ノ ハ ヒ フ ヘ ホ マ
ミ ム メ モ ヤ ユ ヨ ラ リ ル レ ロ ワ ン ゛ ゜
═ ╞ ╪ ╡ ◢ ◣ ◥ ◤ ♠ ♥ ♦ ♣ ● ○ ╱ ╲
╳ 円 年 月 日 時 分 秒
Ok
```

# RIGHT$

**機 能**　文字列の右側から任意の長さの文字列を与えます．

**書 式**　RIGHT$(〈文字列〉, 〈数式〉)

**文 例**　PRINT RIGHT$(A$, 4)

**解 説**

〈数式〉の値は 0 から255までの範囲になければなりません．

〈数式〉の値が 0 ならば，RIGHT$は空の文字列を値とします．

〈数式〉が〈文字列〉の文字数より大きければ，〈文字列〉のすべてを値とします．

参照：LEFT$，MID$

**サンプルプログラム**

```
100 '  文字列（１０文字）の右回転
110  INPUT A$
120  FOR I=1 TO 11
130   PRINT A$
140   B$=RIGHT$(A$,1)
150   A$=B$+MID$(A$,1,LEN(A$)-1)
160  NEXT I
170 END
```

```
run
? 0123456789
0123456789
9012345678
8901234567
7890123456
6789012345
5678901234
4567890123
3456789012
2345678901
1234567890
0123456789
Ok
```

# RND

| | |
|---|---|
| 機　能 | 乱数を与えます. |
| 書　式 | RND〔(<数式>)〕 |
| 文　例 | R＝RND(1) |

解　説

0以上1未満の乱数を与えます.発生する乱数は,RUN及びCLEARが実行されるごとに同系列を取ります.ただし,RANDOMIZE文によってその系列をかえることもできます.

発生される乱数は<数式>の値によって次のように異なります.

<数式> が負の場合——乱数系列を初期化します.
<数式> が0の場合——1つ前に発生した乱数の値をとります.
<数式> が正の場合——次の乱数を発生します.

<数式>が省略された場合は正の値として次の乱数を発生します.

参照：RANDOMIZE

サンプルプログラム

```
100 '  乱数の検定
110 DIM SUM(6)
120 FOR I=1 TO 100
130  DA=INT(RND*6+1)
140  SUM(DA)=SUM(DA)+1
150 NEXT I
160 FOR I=1 TO 6
170  PRINT I;"-";SUM(I);"%"
180 NEXT I


run
 1 - 22%
 2 - 13%
 3 - 19%
 4 - 17%
 5 - 11%
 6 - 18%
Ok
```

# SEARCH　（DISK モード）

機　能　任意の配列変数の要素の中から指定された値を捜し出し，その要素の順位を与えます．

書　式　SEARCH （〈配列変数名〉，〈整数表記〉〔，〈開始ナンバー〉〕〔，〈ステップ値〉〕）

文　例　NUM＝SEARCH(A%，100，0，3)

解　説　〈配列変数名〉で指定された配列変数の要素の中から〈整数表記〉で指定された値を捜し，最初に見つかった要素ナンバーを返します．指定された値がその配列変数内で見つからなかった場合は－1を値として返します．

〈配列変数名〉は整数型の一次元配列でなければならず，またこの配列変数はSEARCH 関数の実行に先だって DIM 文によって宣言されていなければなりません．

〈整数表記〉は配列要素の中から抽出したい値を指定します．もちろんこの値は整数でなければならず，もし実数を指定した場合は整数化してから実行します．

〈開始ナンバー〉と〈ステップ値〉はオプションであり省略可能です．〈開始ナンバー〉には，配列内のデータのどこから捜し始めるかを指定します．省略した場合は配列内データの最初から捜し始めます．ただし，この値は現在 OPTION BASE によって設定されている添字の大きさによって異なります．

〈ステップ値〉を指定した場合はそのステップをもってサーチします．省略した場合は1が採用されすべての要素についてサーチすることになります．

SEARCH はランダムアクセスファイルのインデックス のサーチや GET@，PUT@で用いる配列データのサーチなどに使うことができます．

サンプル
プログラム

```
100 DATA 国語,数学,英語
110 '  出席番号と各科目の点数
120 DATA 1,87,34,56
130 DATA 3,96,85,29
140 DATA 5,85,38,68
150 DATA 2,75,97,54
160 DATA 4,49,95,84
170 '
180 N=5:ITEM=4
190 DIM INDEX%(N*ITEM-1),ITEM$(3)
200 FOR I=1 TO 3:READ ITEM$(I):NEXT I
210 FOR I=0 TO N-1
220  FOR J=0 TO ITEM-1
230   READ INDEX%(I*ITEM+J)
240  NEXT J
250 NEXT I
260 '
270 *ENTRY
280 INPUT "出席番号（１－５）",SN
290 FAC=SEARCH(INDEX%,SN,,ITEM)
300 IF FAC=-1 THEN 280
310 FOR I=1 TO 3
320  PRINT ITEM$(I),INDEX%(FAC+I)
330 NEXT I
340 PRINT:GOTO *ENTRY
```

```
run
出席番号（１－５）1
国語            87
数学            34
英語            56

出席番号（１－５）2
国語            75
数学            97
英語            54

出席番号（１－５）5
国語            85
数学            38
英語            68

出席番号（１－５）
Break in 280
Ok
```

S

# SGN

| | |
|---|---|
| 機 能 | 符号を調べます. |
| 書 式 | SGN(〈数式〉) |
| 文 例 | PRINT SGN(A) |

解 説

〈数式〉が正の場合は1を，〈数式〉が0の場合は0を，〈数式〉が負の場合には－1を，それぞれ与えます.

サンプルプログラム

```
100  SCREEN 0,0:CLS 3
110   LINE(0,100)-(639,100),1
120   X=5
130   FOR I=0 TO 31.42 STEP .05
140    S=SGN(INT(COS(I)*20))
150    ON S+2 GOSUB *PLUS,*ZERO,*MINUS
160    X=X+1
170   NEXT I
180 END
190 *PLUS
200  PSET(X,50),5:RETURN
210 *ZERO
220  LINE(X,50)-(X,149),5:RETURN
230 *MINUS
240  PSET(X,149),5:RETURN
```

# SIN

機能　正弦(サイン)を与えます．

書式　SIN(<数式>)

文例　PRINT SIN(1)

解説　<数式>の値に対する正弦値を与えます．<数式>の単位はラジアンです．

<数式>に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には，単精度の値を与えます．

参照：COS，TAN

サンプルプログラム

```
100 '  ＣＯＳＥＣを求める
110 PI=3.14159
120 F$="cosec(###°)=###.#####"
130 INPUT "角度  ",D
140 R=D MOD 180
150  IF R=0 THEN PRINT "定義出来ません":END
160 C=1/SIN(R/180*PI)
170 PRINT USING F$;D,C
180 END


run
角度  30
cosec( 30°)=  2.00000
Ok
run
角度  60
cosec( 60°)=  1.15470
Ok
```

S

# SPACE$

| | |
|---|---|
| 機　能 | **任意の長さの空白文字列を与えます.** |
| 書　式 | SPACE$(〈数式〉) |
| 文　例 | PRINT SPACE$(10) ; "ABC" |

解　説

〈数式〉の数だけの空白をもった文字列を与えます.その値は0から255までの範囲になければなりません.

参照：SPC

サンプルプログラム

```
100 '-- square curve --
110  FOR I=-6 TO 6
120   A$="|"+SPACE$(ABS(I*I))+"*"
130   PRINT A$
140  NEXT I
150 END

run
|                                    *
|                         *
|                *
|         *
|    *
| *
|*
| *
|    *
|         *
|                *
|                         *
|                                    *
Ok
```

# SPC

**機　能**　任意の数だけ空白を出力します.

**書　式**　SPC(〈数式〉)

**文　例**　PRINT SPC(10); A$

**解　説**　SPC関数はPRINT文やLPRINT文など出力文中でのみ使用することができます.

〈数式〉の値は, −32768から32767まで許されますが, 負の数はすべて0と見なされます. また正の数の場合でも, 現在の水平表示文字数以上の場合は, 〈数値〉をその水平表示文字数で割った余りを値とします.

参照：SPACE$, TAB

サンプルプログラム

```
100 FOR I=0 TO 10
110  PRINT "BASICインタプリタ";SPC(I);"Personal Computer"
120  NEXT I
130 END

run
BASICインタプリタPersonal Computer
BASICインタプリタ Personal Computer
BASICインタプリタ  Personal Computer
BASICインタプリタ   Personal Computer
BASICインタプリタ    Personal Computer
BASICインタプリタ     Personal Computer
BASICインタプリタ      Personal Computer
BASICインタプリタ       Personal Computer
BASICインタプリタ        Personal Computer
BASICインタプリタ         Personal Computer
BASICインタプリタ          Personal Computer
Ok
```

# SQR

機　能　平方根(スクェアールート)を与えます.

書　式　SQR(〈数式〉)

文　例　A=SQR(2)

解　説　〈数式〉で指定された値の平方根を与えます.

〈数式〉に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には単精度の値を与えます.

サンプルプログラム

```
100 '-- Square root table --
110 PRINT " << SQUARE ROOT TABLE >>"
120 FOR N=0 TO 10
130   PRINT "N=";N,"ROOT(N)=";
140   PRINT USING "##.#####";SQR(N)
150 NEXT N

run
 << SQUARE ROOT TABLE >>
N= 0          ROOT(N)= 0.00000
N= 1          ROOT(N)= 1.00000
N= 2          ROOT(N)= 1.41421
N= 3          ROOT(N)= 1.73205
N= 4          ROOT(N)= 2.00000
N= 5          ROOT(N)= 2.23607
N= 6          ROOT(N)= 2.44949
N= 7          ROOT(N)= 2.64575
N= 8          ROOT(N)= 2.82843
N= 9          ROOT(N)= 3.00000
N= 10         ROOT(N)= 3.16228
Ok
```

# STR$

機 能　数値を表す文字列を与えます.

書 式　STR$(<数式>)

文 例　A$=STR$(123)

解 説　<数式>によって指定された値を文字に変換します. <数式>の値は整数型と実数型(倍精度, 単精度)の数値型に有効です.

注意：STRING$関数と区別すること.

参照：VAL, STRING$

サンプルプログラム

```
100 '  ００：００：００から指定した秒数だけ
110 '  経過した時点の時刻を文字列で求める
120  DEF FNST$(A$)=RIGHT$("0"+RIGHT$(A$,LEN(A$)-1),2)
130  INPUT "秒数  : ",SS
140  H=SS¥3600:M=(SS-H*3600)¥60
150  S=SS MOD 60
160  H$=STR$(H):M$=STR$(M):S$=STR$(S)
170  H$=FNST$(H$):M$=FNST$(M$):S$=FNST$(S$)
180  PRINT "00:00:00 から ";SS;"秒 経過した";
190  PRINT " 時刻は ";H$;":";M$;":";S$;" です"
200 END
```

```
run
秒数  : 10000
00:00:00 から  10000 秒 経過した 時刻は 02:46:40 です
Ok
```

# STRING$

機　能　　任意の文字を任意の数だけ与えます.

書　式　　STRING$(〈式〉, ｜〈文字式〉｜)
　　　　　　　　　　　　　　｜〈数式〉　｜

文　例　　A$＝STRING$(10, 〝＋〞)

解　説　　与える文字は，〈文字式〉または〈数式〉によって指定します．〈文字式〉の場合は，与える〈文字列〉の最初の一文字が有効です．〈数式〉の場合は，値が，0から255の範囲に限られ，この値をキャラクタコードとみなし，対応する文字を採用します．

参照：STR$

サンプルプログラム

```
100 INPUT A$
110 L=LEN(A$)
120 PRINT STRING$(L+4,"+")
130 PRINT "+ ";SPC(LEN(A$)+1)+"+"
140 PRINT "+ ";A$;" +"
150 PRINT "+ ";SPC(LEN(A$)+1);"+"
160 PRINT STRING$(L+4,"+")
170 END
```

```
run
? STRING statement sample program
++++++++++++++++++++++++++++++++++++
+                                  +
+ STRING statement sample program +
+                                  +
++++++++++++++++++++++++++++++++++++
Ok
```

# TAB

機　能　現在のカーソルのある行の任意の位置まで空白を出力します.

書　式　TAB(<数式>)

文　例　PRINT TAB(10) ; "BASIC"

解　説　TAB は PRINT 文や LPRINT 文など出力文中のみで使用されます.

<数式>の値は, −32768から32767(0が左端)まで許されますが, 負の数はすべて0と見なされます. また正の数の場合でも, 現在の水平表示文字数以上の場合は, <数値>をその水平表示文字数で割った余りを値とします.

SPC との違いに注意してください.

**注意**：日本語データに対するタブ制御は行えません.

**参照**：SPC

サンプルプログラム

```
100 FOR DEG=0 TO 180 STEP 20
110  RAD=DEG/180*3.14159
120  PRINT "DEG=";DEG;
130  PRINT TAB(15)"SIN(DEG)=";SIN(RAD);
140  PRINT TAB(40)"COS(DEG)=";COS(RAD)
150 NEXT DEG
```

```
run
DEG= 0         SIN(DEG)= 0               COS(DEG)= 1
DEG= 20        SIN(DEG)= .34202          COS(DEG)= .939693
DEG= 40        SIN(DEG)= .642787         COS(DEG)= .766045
DEG= 60        SIN(DEG)= .866025         COS(DEG)= .500001
DEG= 80        SIN(DEG)= .984807         COS(DEG)= .173649
DEG= 100       SIN(DEG)= .984808         COS(DEG)=-.173647
DEG= 120       SIN(DEG)= .866026         COS(DEG)=-.499999
DEG= 140       SIN(DEG)= .642789         COS(DEG)=-.766043
DEG= 160       SIN(DEG)= .342022         COS(DEG)=-.939692
DEG= 180       SIN(DEG)= 2.62155E-06     COS(DEG)=-1
Ok
```

# TAN

**機　能**　正接(タンジェント)を与えます.

**書　式**　TAN(〈数式〉)

**文　例**　A＝TAN(123)

**解　説**

〈数式〉の値に対する正接値を与えます. 〈数式〉の単位はラジアンです.

〈数式〉に倍精度実数が含まれる場合は, 倍精度の値を与えますが, 他の場合には, 単精度の値を与えます.

参照：SIN, COS

**サンプルプログラム**

```
100 '   C O T を 求 め る
110  PI=3.14159
120  F$="cot(###°)=###.#####"
130  INPUT "角 度   ",D
140  R=D MOD 180
150   IF R=0 THEN PRINT "定 義 で き ま せ ん":END
160  C=1/TAN(R/180*PI)
170  PRINT USING F$;D,C
180 END
```

```
run
角度  45
cot( 45°)=  1.00000
Ok
run
角度  30
cot( 30°)=  1.73205
Ok
```

# TIME$

機能　内蔵クロックの時刻を与えます.

書式　1)　TIME$

2)　TIME$＝"hh : mm : ss"

文例　PRINT TIME$

解説　TIME$はシステム変数であり現在の時刻を持っています.

TIME$の形式は,

HH : MM : SS

で示されます.

また, TIME$は次のように新たに時刻を設定することもできます. HH は00から23まで, MM は00から59まで, SS は00から59までの数です.

TIME$＝"12 : 30 : 00"

参照 : DATE$

サンプルプログラム

```
100 '  指定時間後の時刻を求める
110 T$=TIME$
120 PRINT "現在の時刻は  ";T$;"です"
130 INPUT "時間（０－２３）：",H
140 INPUT "分  （０－５９）：",M
150 INPUT "秒  （０－５９）：",S
160 HH=VAL(LEFT$(T$,2))
170 MM=VAL(MID$(T$,4,2))
180 SS=VAL(RIGHT$(T$,2))
190 SS=SS+S:MM=MM+M:HH=HH+H
200 FOR I=0 TO 1
210  MM=MM+SS¥60:SS=SS MOD 60
220  HH=HH+MM¥60:MM=MM MOD 60
230  DD=DD+HH¥24:HH=HH MOD 24
240 NEXT
250 PRINT "求める時刻は"
260  IF DD=1 THEN PRINT "翌日の";
270 PRINT HH;"時  ";MM;"分  ";SS;"秒  です．"
280 END


run
現在の時刻は  22:59:47です
時間（０－２３）：18
分  （０－５９）：15
秒  （０－５９）：2
求める時刻は
翌日の 17時 14分 49秒  です.
Ok
```

# USR

**機能**　**機械語で作られた関数ルーチンを呼び出します.**

**書式**　USR〔〈番号〉〕(〈引数〉)

**文例**　I=USR3(J)

**解説**

ユーザーの作成した機械語関数ルーチン(ユーザー関数)を呼び出します．ユーザー関数は，あらかじめメモリ中に準備されており，またDEF USR文により，その実行開始番地が設定されていなければなりません．

〈番号〉は0から9までの値で，DEF USR文により定義された番号に対応します．〈番号〉が省略された場合には0と解釈されます．

〈引数〉によりBASICから機械語ルーチンへの値の受け渡しをすることができます．

なお，USR関数により機械語ルーチンを呼び出す前に，DEF SEG文によりDEF USR文でUSR関数を定義した時と同じ位置に，セグメントベースが位置付けられていなければなりません．

機械語ルーチンを呼び出す方法としてUSR関数の他にCALL文が用意されています．CALL文では複数の引数を機械語プログラムに受け渡すことができます．

参照：DEF USR，CALL，CLEAR，BLOAD

# VAL

機　能　**文字列の表す数値を与えます．**

書　式　VAL(<文字列>)

文　例　A＝VAL("2")

解　説　〈文字列〉を数値に変換して与えます．もし，〈文字列〉の最初の文字が＋，－，&，または数字でなければ，VALの値は0になります．

ただし，文字列中のスペースは無視します．

参照：STR$

サンプルプログラム

```
100 '  ００：００：００から
110 '  現在の時刻までの秒数を求める
120  T$=TIME$
130  H=VAL(LEFT$(T$,2))
140  M=VAL(MID$(T$,4,2))
150  S=VAL(RIGHT$(T$,2))
160  SS=H*3600+M*60+S
170  PRINT "００：００：００から  ";T$;"  までは";
180  PRINT SS;"秒です"
190 END

run
００：００：００から  23:07:27  までは 83247 秒です
Ok
```

U

V

# VARPTR

**機　能**　変数の値が格納されているメモリ番地，ファイルに割り当てられているファイルコントロールブロックの開始番地を与えます．

**書　式**
1)　VARPTR(〈変数名〉〔,〈機能〉〕)
2)　VARPTR(#〈ファイル番号〉〔,〈機能〉〕)

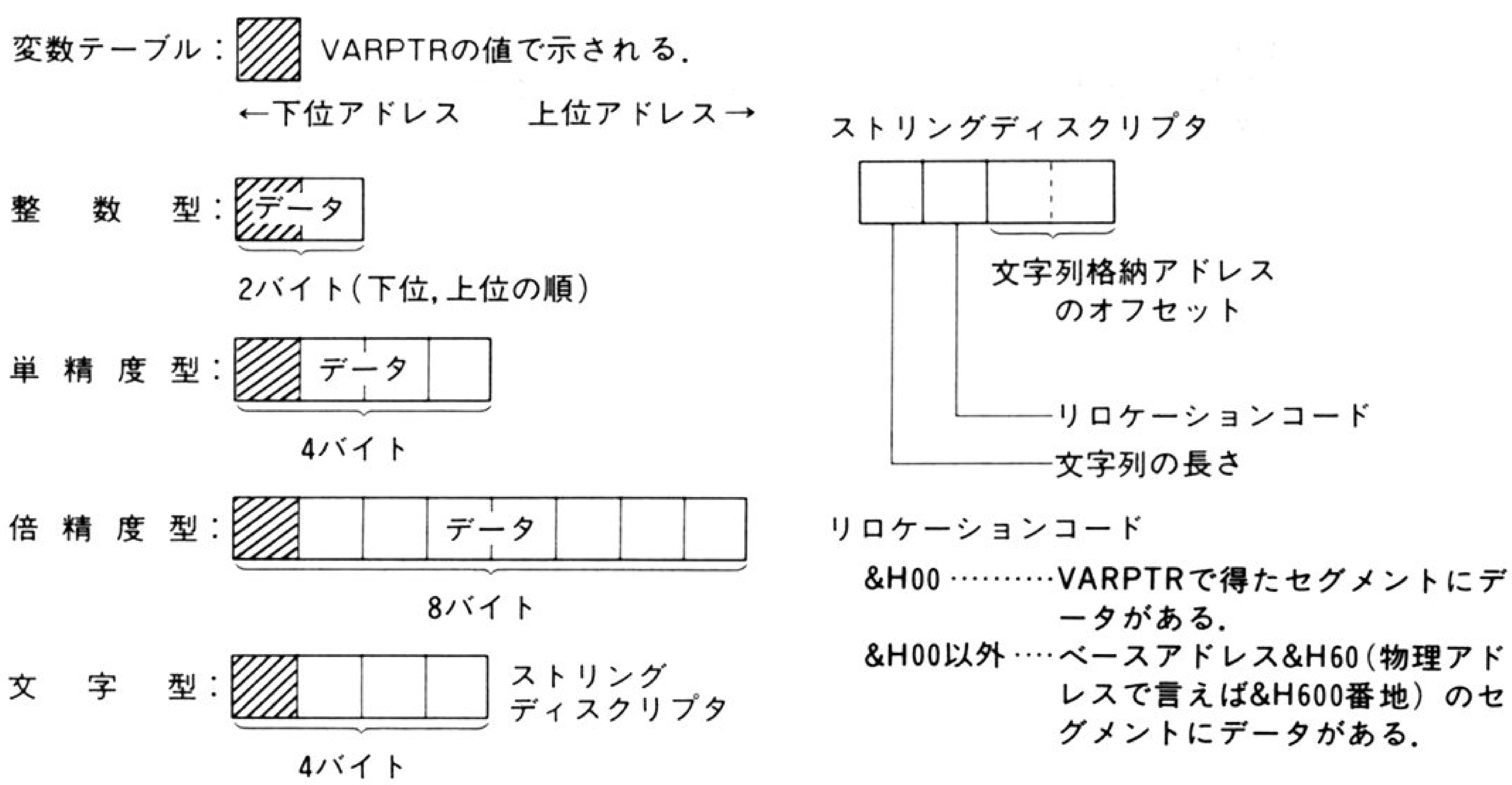
**文　例**
1)　PRINT HEX$(VARPTR (A, 1))
　　PRINT HEX$(VARPTR (A))
2)　BSG%=VARPTR(#1, 1)
　　BAD%=VARPTR(#1)

**解　説**　**書式1)**　〈変数名〉で指定した変数あるいは配列要素のデータが格納されている領域のメモリ番地についてベースアドレスあるいは相対アドレスを与えます．〈機能〉の指定は，0あるいは0以外の整数で次の意味を示します．

0……相対アドレスを返す．

1……ベースアドレスを返す．

なお，〈機能〉を省略した場合は，0が指定されたものとみなされます．



**書式２）** 指定した〈ファイル番号〉に割り当てられているファイルコントロールブロックの開始番地について，ベースアドレスあるいは相対アドレスを与えます．〈機能〉については，変数の場合と同様に値を指定します．
ファイルに割り当てられている入出力バッファの開始番地は，このファイルコントロールブロックの33バイト目からの２バイトに格納されています．

# VIEW

**機　能**　現在のビューポートの設定位置を与えます．

**書　式**　VIEW(<機能>)

**文　例**　SX1＝VIEW(0)

**解　説**　VIEW 文により設定されている現在のビューポートの位置($Sx_1$，$Sy_1$)－($Sx_2$，$Sy_2$)を返す関数です．<機能>は 0 から 3 までの値をとり，それぞれの値のときにとる関数値の意味は次のとおりです．

0：ビューポートの左上の頂点のX座標（$Sx_1$）
1：ビューポートの左上の頂点のY座標（$Sy_1$）
2：ビューポートの右下の頂点のX座標（$Sx_2$）
3：ビューポートの右下の頂点のY座標（$Sy_2$）

VIEW 関数は，ディスプレイ画面上のビューポートの位置を返す関数ですから，返される値は，オリジナルスクリーン座標であることに注意してください．

参照：VIEW，MAP，WINDOW，WINDOW 関数

**サンプルプログラム**

```
100 SCREEN 0,0
110 GOSUB *VIEWINFO
120 WINDOW(-100,-100)-(100,100)
130 GOSUB *VIEWINFO
140 VIEW(100,50)-(300,150)
150 GOSUB *VIEWINFO:END
160 *VIEWINFO
170  PRINT "SX1 :";VIEW(0),
180  PRINT "SY1 :";VIEW(1)
190  PRINT "SX2 :";VIEW(2),
200  PRINT "SY2 :";VIEW(3):PRINT
210 RETURN


run
SX1 : 0       SY1 : 0
SX2 : 639     SY2 : 199

SX1 : 0       SY1 : 0
SX2 : 639     SY2 : 199

SX1 : 100     SY1 : 50
SX2 : 300     SY2 : 150

Ok
```

# WINDOW

機　能　**現在のウィンドウの設定位置を与えます.**

書　式　WINDOW(<機能>)

文　例　WY1＝WINDOW(1)

解　説　WINDOW 文により設定されている現在のウインドウの位置($Wx_1$, $Wy_1$)－($Wx_2$, $Wy_2$)を返す関数です. <機能>は, 0から3までの値をとり, それぞれの値のときにとる関数値の意味は次のとおりです.

0：ウィンドウの左上の頂点のX座標（$Wx_1$）
1：ウィンドウの左上の頂点のY座標（$Wy_1$）
2：ウィンドウの右下の頂点のX座標（$Wx_2$）
3：ウィンドウの右下の頂点のY座標（$Wy_2$）

WINDOW 関数の返す値は, ワールド座標であることに注意してください.

参照：VIEW, VIEW 関数, MAP, WINDOW

サンプルプログラム

```
100 SCREEN 0,0
110 GOSUB *WINDOWINFO
120 VIEW(100,50)-(300,150)
130 GOSUB *WINDOWINFO
140 WINDOW(-100,-100)-(100,100)
150 GOSUB *WINDOWINFO
160 END
170 *WINDOWINFO
180  PRINT "WX1 :";WINDOW(0),
190  PRINT "WY1 :";WINDOW(1)
200  PRINT "WX2 :";WINDOW(2),
210  PRINT "WY2 :";WINDOW(3):PRINT
220 RETURN


run
WX1 : 0        WY1 : 0
WX2 : 639      WY2 : 199

WX1 : 100      WY1 : 50
WX2 : 300      WY2 : 150

WX1 :-100      WY1 :-100
WX2 : 100      WY2 : 100

Ok
```

W

# APPENDIX

# A. エラーメッセージ一覧

| エラーメッセージ | コード | 意　　味 |
| --- | --- | --- |
| Bad allocation table | 69 | ディスケット内部のFATが壊れている. |
| Bad drive number | 70 | ドライブ指定が誤っている. システムにつながっていないドライブを指定した. |
| Bad file name | 56 | ファイル名の指定がまちがっている. |
| Bad file number | 52 | ファイル番号が正しくない（同時オープン数を超えたオープン要求）. |
| Bad track/ sector | 71 | トラック, セクタ番号の指定が誤っている. (DSKO$, DSKI$) |
| Can't continue | 17 | CONT命令によって実行を再開することができない. |
| Deleted record | 72 | 消去済みのレコードをアクセスしようとした. |
| Direct statement in file | 57 | アスキー形式のファイルを読み込む際にダイレクトステートメントが存在した. |
| Disk full | 68 | ディスク上に書き込むスペースがない. |
| Disk I/O error | 64 | ディスクとの入出力中にエラーが発生した. 致命的なエラーであり回復させることはできない. |
| Disk offline | 62 | ディスクがセットされていないドライブをアクセスした. |
| Division by Zero (/0) | 11 | 0による割り算が実行されて, その結果の値がオーバーフローした. |
| Duplicate Definition | 10 | 配列またはユーザー関数を二重定義しようとした. |
| Duplicate label | 31 | 同じラベル名が2つ以上存在している. |
| Feature not available | 33 | 利用不可能な機能を指定した. |

| エラーメッセージ | コード | 意　　味 |
|---|---|---|
| FIELD overflow | 50 | FIELD文において256バイト以上の大きさの領域を指定した． |
| File already exist | 65 | NAME文によって変更しようとしたファイル名が既に存在している． |
| File already open | 54 | 既にオープンされているファイルに対してOPEN，KILLなどを実行した． |
| File not found | 53 | 指定されたファイルが見つからない． |
| File not open | 60 | オープンされていないファイルを参照しようとした． |
| File write protected | 61 | 書き込み禁止属性が付けられているファイルに書き込もうとした． |
| FOR without NEXT | 26 | FOR～NEXTが正しく対応していない（FORが多い）． |
| Illegal direct | 12 | ダイレクトモードで使用できない文を実行しようとした． |
| Illegal function call | 5 | ステートメントおよび関数において，機能の呼び方が誤っている． |
| Illegal operation | 74 | 不正なキーボード操作をした． |
| Input past end | 55 | ファイル中のすべてのデータを読み尽した後に，さらに入力文が実行された． |
| システム予約 | 51 | ―――――――――― |
| Line buffer overflow | 23 | 1行で入力できる文字の範囲を越えて入力が行われた． |
| Missing operand | 22 | ステートメント中必要なパラメータが指定されていない． |
| NEXT without FOR | 1 | FOR～NEXTが正しく対応していない．(NEXTが多い) |

| エラーメッセージ | コード | 意　　味 |
|---|---|---|
| No RESUME | 19 | エラー処理ルーチンの中にRESUME文がなく，プログラムの実行が継続できない． |
| Out of DATA | 4 | 読むべきデータがないのにREAD文が実行された． |
| Out of memory | 7 | メモリ容量が足りなくなった（プログラムが大きすぎる，またはスタックを消費し尽くした等）． |
| Out of string space | 14 | 文字列を格納するメモリ領域がなくなった． |
| Overflow (OV) | 6 | 演算結果や入力された数値が，許される範囲を越えた． |
| Rename across disks | 73 | NAME文において，異なるドライブ間でリネームしようとした． |
| RESUME without error | 20 | エラー処理ルーチンに制御を移さなかったのに，RESUME文がある． |
| RETURN without GOSUB | 3 | GOSUB～RETURNが正しく対応していない．(RETURNが多い) |
| Sequential after PUT | 58 | PUT文実行後，シーケンシャルファイルをアクセスしようとした． |
| Sequential I/O only | 59 | シーケンシャル入出力以外は行ってはならない． |
| String formula too complex | 16 | 文字式が複雑すぎる．（カッコのネスティングが深すぎるなど） |
| String too long | 15 | 1つの文字変数内の文字数が255文字を越えている． |
| Subscript out of range | 9 | 配列変数の添字の値が規定の範囲から外れている． |
| Syntax error | 2 | 文の記述が文法通りになっていない． |
| Tape read error | 27 | テープからの読み込み時にエラーが発生した． |

| エラーメッセージ | コード | 意　　味 |
|---|---|---|
| Type mismatch | 13 | 式の左辺・右辺，関数の引数などにおいて変数の型が一致していない． |
| Undefined label | 32 | 定義されていないラベル名で参照した． |
| Undefined line number | 8 | 飛び先として必要とされる行番号（GOTO, GOSUBなどの飛び先）が存在しない． |
| Undefined user function | 18 | DEF FN文によって定義されていないユーザー関数を引用しようとした． |
| Unprintable error | 21 | メッセージの定義されていないエラー． |
| WEND without WHILE | 30 | WHILE～WENDが正しく対応していない．(WENDが多い) |
| WHILE without WEND | 29 | WHILE～WENDが正しく対応していない．(WHILEが多い) |

# B. コントロールコード一覧

| キャラクタコード | 対応するキー | 機　　　能 |
|---|---|---|
| 1 | CTRL-A | ヘルプキーと同じ |
| 2 | CTRL-B | 1つ前のワードへ戻る |
| 3 | CTRL-C | 実行の停止，ストップキーと同じ |
| 4 | CTRL-D | 1ワード削除 |
| 5 | CTRL-E | 1行削除 |
| 6 | CTRL-F | 1つ先のワードへ進む |
| 7 | CTRL-G | スピーカを鳴らす* |
| 8 | CTRL-H | バックスペース* |
| 9 | CTRL-I | 水平タブ．タブキーと同じ* |
| 10 | CTRL-J | ラインフィード*．インサートモードで2行に分割 |
| 11 | CTRL-K | ホームポジション* |
| 12 | CTRL-L | テキスト画面クリア* |
| 13 | CTRL-M | キャリッジリターン* |
| 15 | CTRL-O | 画面への表示の無効／有効を切り替える |
| 18 | CTRL-R | インサートモード |
| 19 | CTRL-S | 表示を一時停止する |
| 21 | CTRL-U | 1行キャンセル |
| 24 | CTRL-X | 1行の最後へ進む |
| 28 | → | カーソルを右へ移動* |
| 29 | ← | カーソルを左へ移動* |
| 30 | ↑ | カーソルを上へ移動* |
| 31 | ↓ | カーソルを下へ移動* |

備考　*印がついているものは，対応するコードを画面に出力することにより，機能欄の動作を実行させることができます．

# C. キャラクタコード

上位 4 ビット→

| 下位4ビット↓ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | $D_E$ | | 0 | @ | P | | p | | | | ー | タ | ミ | | |
| 1 | $S_H$ | $D_1$ | ! | 1 | A | Q | a | q | | | 。 | ア | チ | ム | | 円 |
| 2 | $S_X$ | $D_2$ | " | 2 | B | R | b | r | | | 「 | イ | ツ | メ | | 年 |
| 3 | $E_X$ | $D_3$ | # | 3 | C | S | c | s | | | 」 | ウ | テ | モ | | 月 |
| 4 | $E_T$ | $D_4$ | $ | 4 | D | T | d | t | | | 、 | エ | ト | ヤ | | 日 |
| 5 | $E_Q$ | $N_K$ | % | 5 | E | U | e | u | | | ・ | オ | ナ | ユ | | 時 |
| 6 | $A_K$ | $S_N$ | & | 6 | F | V | f | v | | | ヲ | カ | ニ | ヨ | | 分 |
| 7 | $B_L$ | $E_B$ | ' | 7 | G | W | g | w | | | ァ | キ | ヌ | ラ | | 秒 |
| 8 | $B_S$ | $C_N$ | ( | 8 | H | X | h | x | | | ィ | ク | ネ | リ | ♠ | |
| 9 | $H_T$ | $E_M$ | ) | 9 | I | Y | i | y | | | ゥ | ケ | ノ | ル | ♥ | |
| A | $L_F$ | $S_B$ | * | : | J | Z | j | z | | | ェ | コ | ハ | レ | ♦ | |
| B | $H_M$ | $E_C$ | + | ; | K | [ | k | { | | | ォ | サ | ヒ | ロ | ♣ | |
| C | $C_L$ | → | , | < | L | ¥ | l | ¦ | | | ャ | シ | フ | ワ | ● | |
| D | $C_R$ | ← | − | = | M | ] | m | } | | | ュ | ス | ヘ | ン | ○ | |
| E | $S_O$ | ↑ | . | > | N | ∧ | n | ～ | | | ョ | セ | ホ | ゛ | | |
| F | $S_I$ | ↓ | / | ? | O | _ | o | | | | ッ | ソ | マ | ゜ | | |

# D. 誘導関数

関数として用意していない三角関数のうち，いくつかは，用意された関数を使って作り出すことができます．以下の数式を参考にしてください（誤差の範囲に注意が必要です）．

| 目的とする関数 | 組み込み関数からの誘導式 |
|---|---|
| セカント | SEC(X)=1/COS(X) |
| コセカント | CSC(X)=1/SIN(X) |
| コタンジェント | COT(X)=1/TAN(X) |
| アークサイン | ARCSIN(X)=ATN(X/SQR(−X * X+1)) |
| アークコサイン | ARCCOS(X)=−ATN(X/SQR(−X * X+1))+1.5708 |
| アークセカント | ARCSEC(X)=ATN(SQR(X * X−1)+(SGN(X)−1) * 1.5708 |
| アークコセカント | ARCCSC(X)=ATN(1/SQR(X * X−1))+(SGN(X)−1) * 1.5708 |
| アークコタンジェント | ARCCOT(X)=−ATN(X)+1.5708 |
| ハイパーボリック・サイン | SINH(X)=(EXP(X)−EXP(→X))/2 |
| ハイパーボリック・コサイン | COSH(X)=(EXP(X)+EXP(−X))/2 |
| ハイパーボリック・タンジェント | TANH(X)=−EXP(−X)/(EXP(X)+EXP(−X)) * 2+1 |
| ハイパーボリック・セカント | SECH(X)=2/(EXP(X)+EXP(−X)) |
| ハイパーボリック・コセカント | CSCH(X)=2/(EXP(X)−EXP(−X)) |
| ハイパーボリック・コタンシェント | COTH(X)=EXP(−X)/(EXP(X)−EXP(−X)) * 2+1 |
| ハイパーボリック・アークサイン | ARCSINH(X)=LOG(X+SQR(X * X+1)) |
| ハイパーボリック・アークコサイン | ARCCOSH(X)=LOG(X+SQR(X * X−1)) |
| ハイパーボリック・アークタンジェント | ARCTANH(X)=LOG((1+X)/(1−X))/2 |
| ハイパーボリック・アークセカント | ARCSECH(X)=LOG((SQR(−X * X+1)+1)/X) |
| ハイパーボリック・アークコセカント | ARCCSCH(X)=LOG((SGN(X) * SQR(X * X+1)+1)/X) |
| ハイパーボリック・アークコタンジェント | ARCCOTH(X)=LOG((X+1)/(X−1))/2 |

# E. 予約語

これらの予約語を変数名，ラベル名として使用することはできません．

| | | | |
|---|---|---|---|
| ABS | CONT | EOF | INPUT |
| AKCNV$ | COPY | EQV | INPUT$ |
| ALLOC | COS | ERASE | INSTR |
| AND | CSNG | ERL | INT |
| ASC | CSRLIN | ERR | IRESET |
| ATN | CVD | ERROR | ISET |
| ATTR$ | CVI | EXP | JIS$ |
| AUTO | CVS | FIELD | KACNV$ |
| BEEP | DATA | FILES | KANJI |
| BLOAD | DATE$ | FIX | KEXT$ |
| BSAVE | DEF | FN | KEY |
| CALL | DEFDBL | FOR | KILL |
| CDBL | DEFINT | FRE | KINPUT |
| CHAIN | DEFSNG | GET | KINSTR |
| CHR$ | DEFSTR | GOTO | KLEN |
| CINT | DELETE | GO TO | KMID$ |
| CIRCLE | DELIM | GOSUB | KNJ$ |
| CLEAR | DIM | HELP | KPLOAD |
| CLOSE | DRAW | HEX$ | KTYPE |
| CLS | DSKF | IEEE | LEFT$ |
| CMD | DSKI$ | IF | LEN |
| COLOR | DSKO$ | IFC | LET |
| COM | EDIT | IMP | LFILES |
| COMMON | ELSE | INKEY$ | LINE |
| CONSOLE | END | INP | LIST |

LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MAIL
MAP
MERGE
MID$
MKD$
MKI$
MKS$
MOD
MON
MOTOR
NAME
NEXT
NEW
NOT
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
PAINT
PEEK
PEN
PLAY
POINT
POKE
POLL
POS
PPOLL
PPR
PRESET
PRINT
PSET
PUT
RANDOMIZE
RBYTE
READ
REM
REN
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROLL
RSET
RUN
SAVE
SCREEN
SEARCH
SEG
SET
SGN
SIN
SPACE$
SPC
SQR
SRQ
STATUS
STEP
STOP
STR$
STRING$
SWAP
TAB
TAN
THEN
TIME$
TIMEOUT
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VIEW
WAIT
WBYTE
WEND
WHILE
WIDTH
WINDOW
WRITE
XOR

たしかな技術で世界をむすぶ

NEC

PCH32
A9715111